공학박사 학위논문

모바일 기기에서 인간 시각 시스템을 고려한 그래픽 품질 개선 및 전력 소모 최적화

Enhancing graphics quality and optimizing power consumption considering the human visual system in mobile devices

2016년 2월

서울대학교 대학원

전기·컴퓨터공학부

김민규

요 약

최근까지 GPU의 하드웨어가 눈에 띄게 발전하고 있지만, 아직도 60fps를 만족하면서 높은 품질의 그래픽 요구사항을 만족하기 어렵다. 또한 최근 높은 해상도의 요구사항은 전력 소모와 온도 문제 관점에서도 매우 어려운 문제이다. GPU의 전력 소모는 GPU의 연산량과 정비례하기 때문에, 사람의 인지 능력 관점에서 이득이 없음에도 불구하고, 고정된 높은 해상도와 높은 프레임 속도로 인한 GPU 높은 연산량은 의미가 없다. 본 논문에서는 사람의 인지 능력을 고려한 GPU 연산량을 줄이는 새로운 방법들을 제안한다. 사람의 인지 능력을 고려한 GPU 연산량을 줄이는 시작 단계로, 전력 소모의 주요 요인들을 상용화된 LG G3 모바일 기기로 분석한다. 이 과정을 통해 모바일 GPU의 전력 소모의 3 가지 주요 요인인 해상도, 프레임 속도 그리고 데이터 중복성에 대해 분석한다. 이러한 주요 요인들을 기반으로 사람의 인지 능력을 고려한 새로운 렌더링 기법들을 통해 연산량을 효과적으로 절감하는 기법들을 제안한다.

첫째로 해상도 관점에서 GPU에서의 해상도 변경 기반 연산량 감소 기법들에 대해 제안한다. 최근의 연구들은 사람의 인지능력과 콘텐츠의 특성을 반영하지 못하여, 그래픽 결점이 지속적으로 관찰된다. 기존 연구들과는 다르게, 제안하는 동적 렌더링 화질 개선 스케일링 (Dynamic Rendering Quality Scaling: DRQS)은 최소한의 추가비용으로 변환 행렬을 활용한 프레임 간 변화량을 이용하여 해상도 조절 및 품질 개선 스케일링을 통해 성능을 최대 38%까지 개선한다. 또한 저 사양 그래픽스 응용프로그램의 경우에서는 사람의 인지 능력관점에서 그래픽 품질의 감소 없이 GPU의 연산량을 24%까지 줄인다.

둘째로 프레임 보간 기법을 활용한 그래픽 품질 향상 기법들에 대해서 제안한다. 최근의 프레임 보간 방식은 모션 보상 기반의 알고리즘 기반으로 중간프레임을 생성하기 때문에 요구되는 높은 비용은 모바일에서 적용할 수 없다. 이러한 문제를 개선하기 위해, 새로운 접근 방식인 GPU의 타일 렌더링을 이용한 중간 프레임 전달 방식의 프레임 보간 기법은 지연과 추가적인 높은 비용 없이 중간 프레임을 생성한다. 제안하는 기법을 통해 기존의 연구들 대비 시스템 관점에서 절반의 연산 비용으로 사람의 인지 능력 관점에서 동등한 그래픽 품질을 얻을 수 있다.

마지막으로 가장 최근에 발표된 OpenGL ES 3.0에서 제안된 기술인 Multi render target(MRT) 기술을 재사용 관점에서 최적화하기 위한 방법을 제안한다. MRT 는 지연 쉐이딩을 통한 복잡한 라이팅 연산을 효율적으로 처리하기 위해 많이 사용되는 기술이다. 하지만, 한꺼번에 렌더 타깃에 렌더링을 해야 하기 때문에 큰 메모리 대역폭을 요구한다, 이러한 문제는 제한된 모바일 환경에서는 큰 장애이다. 이 문제를 개선하기 위해 시간적 중복성을 이용한 데이터 재사용을 통해 이미 쓰인 렌더 타깃의 데이터를 선택적으로 재사용하여 GPU의 연산량 및 메모리 사용을 감소시킨다. 실험을 통해 사람의 인지 능력 과점에서 그래픽 품질을 유지하면서 18%의 시스템 레벨의 전력 소모 감소를 얻을 수 있다.

주요어 : GPU 실시간 렌더링, 모바일 GPU, 프레임 속도 증가 기법, 동적 렌더링, 멀티 렌더 타깃, 데이터 재사용, GPU 소모 전력 최적화, 화질 기반 스케일링

학 번 : 2012-30931

목 차

그림 목차

표 목차

제 1 장

서 론

1.1 연구 목적

휴대 기기의 진화는 [52] 어떤 기술보다 빠르게 변화하고 있다. 특히, 스마트 폰과 태블릿은 사진을 찍고, 게임을 하며 그리고 E-mail을 읽으며, 웹 브라우징과 Social network 서비스를 위한 주요 기기가 되고 있다. 모바일 컴퓨팅을 위한 다양한 기술발전과 System-on-chip(SoC) 설계의 발전으로 최근 발표되는 스마트 폰은 과거 Desktop PC에서 가능했던 다양한 시나리오들이 가능한 수준까지 왔다. 모바일 기기에서도 사용자 경험관점에서 중요한 사항 중 하나는, 사용자에게 실시간 컴퓨팅 환경을 제공해야 한다. 이 관점에서 모바일 그래픽스 하드웨어와 소프트웨어의 발전은 사용자 경험 관점에서 시각적으로 뛰어난 그래픽 제공과 함께 빠른 응답속도 제공을 목적으로 한다.

요즘 상용화되는 모바일 기기들은 최소한 기본적으로 1초당 60장

의 프레임(60Hz)의 디스플레이를 지원하며, 동시에 FHD(1920x 1080) 화질을 실시간으로 지원 가능해야 한다. 최근에는 QHD(2560 x1440) 뿐 아니라 UHD(3840x2160) 의 지원까지도 고려하고 있다. 실시간 기준의 그래픽 품질과 사용자 환경을 제공하기 위해서는 모바일 GPU는 1초에 60장의 프레임을 렌더링을 해야 하는 성능을 보장해야 한다. 그래픽스 관점에서 다시 말하면, FHD 기준 한 장의 그림을 고려했을 때, 실시간 처리를 위해 1920x1080 픽셀 수만큼 높은 프레임 속도로 복잡한 3D 그래픽스 연산을 해야 한다. 또한 응용 프로그램 관점에서 보면, 각각의 응용 프로그램은 각기 다른 컴퓨팅 연산 양을 요구하며, 이 모든 응용 프로그램을 실시간으로 지원해야 한다는 관점에서, 최근 Benchmark 기준으로 약 4 Giga pixels/second 의 픽셀 처리 성능을 만족해야 한다[69, 53, 77].

하지만, 현 수준의 모바일 GPU는 실시간 환경 기준의 Benchmark 성능을 만족시키지 못하고 있으며[83], 비록 성능을 만족할 수 있다 하더라도, 급격한 소모 전력을 초래한다. 이로 인하여, 모바일의 제한된 전력 소모 관점에서 성능의 제한 및 발열 등 많은 문제가 야기 된다. 지금까지도 저전력 모바일 환경을 위해 하드웨어적 저전력 설계 기술과 GPU의 연산량을 줄이는 소프트웨어적 기술들이 많은 연구가 되고 있음에도 불구하고, 모바일 기기에서 작은 전력량으로 사용자 경험 측면에서 높은 수준의 그래픽 품질과 응답성을 보장하는 것은 쉬운 일이 아니다.

본 논문의 목적은 사용자의 인지 능력 기반으로 품질의 열화를

인지하지 못하는 범위 내에서 그래픽 품질을 조절하여 GPU의 소모 전력을 최적화하고 동시에 그래픽 서비스 품질을 향상 시키는 효과적인 방법을 제안하는 것이다. 적은 추가 비용으로 내장형 그래픽 프로세서의 에너지 효율을 향상시키고, 사용자에게 높은 응답성과 높은 품질의 그래픽 품질을 보장한다. 이를 위해 주어진 모바일 GPU의 하드웨어의 성능의 한계를 하드웨어와 소프트웨어적 접근을 통해 사용자의 인지 능력을 고려하여 문제를 해결한다.

1.2 연구 공헌

본 논문에서는 모바일 GPU의 전력 소모 관점에서 제어 가능한 인자들을 확인하고 제어가능 한 인자 중 치명 인자를 도출하여 각각 인자들을 통한 GPU 연산량 감소 기법을 제안한다. 기존 연구들과는 다르게, 사용자 경험 측면에서 사용자가 인지할 수 있는 능력을 기준으로 하여 인자들의 조절을 통해 GPU의 연산량을 줄이고 곧 GPU의 전력 소모와 발열을 줄이는 효율적이고 새로운 기법들을 제안한다. 모바일 GPU의 전력 소모를 위해 제어 가능한 전력 소모 관점에서의 치명 인자를 도출하고 각각 인자들의 활용 및 최적화를 통한 GPU 연산량 감소 기법을 제안한다.

첫째 주요 인자로, 본 논문에서 GPU 전력 감소를 위해 해상도의 조절을 통해 GPU 연산량을 줄이는 프레임간의 변화량 기반 가변 해상도 처리 및 화질 최적화 기법[3, 6]을 제안한다.

- 해상도 조절 기반 GPU 연산량을 줄이기 위한 최근까지 연구된 다양한 기법들과 각각의 한계점을 분석하였다.
- 프레임 간 변화량을 측정하기 위해 추가적인 비용이 많이 드는 기존의 복잡한 방식을 극복하고, GPU에서 처리하는 행렬 모델 뷰 프로젝션 행렬(Model view projection matrix)을 이용하여 아주 적은 비용으로 프레임 간 변화량 계산 기법을 제안하였다.
- 사용자 경험 측면에서 사람의 인지 능력을 고려하여 높은 해상도에서 낮은 해상도의 변환을 위한 시각 시스템(Human visual system)[10] 기반의 알고리즘을 제안하였다. 프레임 간 이동하는 빠른 물체나 카메라의 시점 이동은 사람의 눈에 뿌옇게 보이는 현상이 있다 [36]. 이러한 현상을 활용하여, 빠르게 이동하는 물체나 카메라의 시점의 변화가 있을 경우, 낮은 해상도로 렌더링하여 GPU 연산량의 최적화 기법을 제안하였다.
- 높은 해상도에서 낮은 해상도로 렌더링하는 경우 화질 열화가 발생할 수 있다. 대표적으로, 프레임 안에 포함되어 있는 글자나 경계 부분이 뭉개지는 현상이 심화되어 나타날 수 있다. 이러한 부분에 대해 화질 최적화 기법을 통해 열화를 최소화 하는 방안을 제안하였다.
- 위의 프레임 간 변화량 기반 가변 해상도 기법을 통해 널리 사용되는 응용프로그램들을 통해 GPU 연산량의 감소를 확인하였으며, 동시에 사람의 인지 능력 측면에서 높은 해상도로만 렌더링하는 경우와 비교하여 화질 열화를 거의 느낄 수 없음을 증명하였다.

- 추가적으로, 위의 기법을 통한 GPU 연산량의 감소는 동일한 GPU하드웨어 에서 1초에 60장 이하로 렌더링하는 연산량이 매우 큰 응용프로그램의 경우에 끊김 현상(flickering)의 개선을 통해 보다 높은 그래픽 품질을 얻을 수 있다. GPU 연산량의 감소는 동일한 조건에서 자연스러운 프레임 속도의 증가 효과 얻을 수 있기 때문이다.

둘째 주요 인자로, 본 논문에서 동일한 GPU 연산량으로 프레임 속도 증가를 통한 그래픽 품질 개선 및 소모전력 최적화를 위해 새로운 타일 기반 GPU를 활용한 중간 프레임 전달 기법[34] 제안하였다.

- 최근까지 연구된 프레임 속도 증가 기법들 중 가장 널리 사용되는 프레임 보간(Frame interpolation) 기술들을 비교 분석하고, 모바일의 제한된 환경에서 연구된 기술들이 가지는 한계점을 분석하였다.
- PC 기반의 GPU 렌더링 기술인 직접 모드 렌더링(Immediate mode rendering) 방식과 저전력 모바일 환경에 적합하도록 적용한 타일 기반 렌더링(Tile based rendering) 기술을 비교 분석하였다.
- 타일 렌더링 기반의 GPU에서 적용 가능한 프레임 속도 증가 기법을 통해 모바일 환경에서 하드웨어 성능 제약으로 인한 끊김 현상을 개선하였다. 추가적으로 최근 연구되는 VR[42] 등과 같은 기술 구현을 위해 기존의 하드웨어 성능의 제한을 극복 하였다.

- 타일 렌더링 기반 우선순위 타일 업데이트 및 합성 알고리즘을 통해 두 장의 연속된 프레임에서 중간 프레임을 최소한의 비용으로 생성하는 기법을 제안하였다.
- 타일 우선순위 기반 합성을 통한 중간 프레임을 생성에 있어서 발생할 수 있는 화질 결함을 최소화하기 위해 프레임 간 동적인 부분을 찾기 위한 최소 비용의 경계검사 기반 동적 타일 검출 알고리즘 제안하였다.
- 널리 사용되는 다양한 응용 프로그램들의 실험을 통해서, 제안된 방법을 통해 프레임 속도를 최대 2배까지 증가 시키는 결과와 동시에 제안된 기법이 시각 시스템 기준으로 화질 열화를 인지하지 못하는 범위로 유지 가능함을 증명하였다.
- 고정된 프레임 속도의 모바일 기기에서는 절반의 GPU 연산량만으로 기존의 프레임 속도를 달성이 가능하다. 다시 말해 동일한 프레임 속도를 유지 하면서 GPU의 연산량과 전력 소모는 절반 가까이 감소 가능함을 증명하였다.

마지막으로 GPU의 전력 감소를 위한 주요 인자인 데이터 재사용을 통한 기법이다. 가장 최근 발표된 OpenGL ES 3.0[62] 의 멀티 렌더 타깃(Multi Render Target)[64] 을 활용한 사람의 인지능력 관점에서 화질 열화를 최소화 하며 데이터 재사용을 통한 GPU의 메모리 대역폭 최적화 기법을 제안한다.

- GPU의 그래픽 연산 처리 관점에서 메모리 대역폭을 감소하기 위한 최근까지 연구된 기술을 분석하였다. 데이터 재사용의 기술

에 적용된 기본 개념을 바탕으로 최근 발표된 OpenGL ES 3.x 에서 적용할 방법을 제안하였다.

- OpenGL ES 3.x[62] 가 발표되면서 추가된 새로운 기술인 복수 렌더 타깃(Multi-render target)[64] 기술에 대해서 설명하고, 모바일 환경에서의 구현 관점에서 기술이 가지고 있는 한계성을 분석하였다.
- 각각의 렌더 타깃의 재사용으로 인한 발생할 수 있는 화질 열화와 메모리 대역폭 측면에서 감소를 분석하였다. 분석 내용 기반으로 사람의 인지 능력 관점에서 성능의 저하를 인지 할 수 없는 범위 내에서 복수 렌더 타깃 기술을 위한 데이터를 재사용 새로운 기법에 대해서 제안하였다.
- 최근 발표된 그래픽 응용 프로그램 중, 복수 렌더 타깃을 가장 많이 활용하며, 메모리 사용을 가장 많이 하는 응용 프로그램을 실험 벡터로 선정하였다. 제안된 방법을 통해 얻을 수 있는 메모리 대역폭 감소와 사람의 인지적 관점에서의 화질의 열화를 최소화하여 제안된 방법의 유효성을 증명하였다.

1.3 논문 구성

하기와 같이 본 논문의 내용을 구성한다. 다음 2장에서는 연구 배경을 설명하고 모바일 그래픽스 환경에서의 전력 소모를 분석하여 전력 소모의 주요 요인들을 정의 한다. 전력 소모의 주요 요인들 중 소모 전력 감소를 위한 모바일 그래픽스 환경에서 최적화 가능한

인자와 치명적인 주요 인자들을 분석한다. 치명적인 주요 인자들 각각의 최적화를 통해 문제 해결 방향을 제안한다.

본문에서는 먼저 2장에서 정의한 GPU의 성능과 에너지 효율을 높이기 위한 주요 인자들인 해상도, 프레임 속도 그리고 데이터 재사용 순으로 각각의 관점에서 최근 연구된 기술들과 한계와 문제점을 분석한다. 각 주요 인자들의 한계와 문제점을 해결하기 위해 본 논문의 주제인 사람의 인지 능력을 고려한 저전력 모바일 기기를 위한 그래픽 품질 개선 및 소모 전력 최적화 기법을 제안한다. 3장에서는 해상도 조절을 통한 GPU연산 및 전력 소모를 줄이기 위한 연구들에 대해서 살펴보고, 가변 해상도 기반으로 저전력 모바일 환경을 위한 해결 방안인, 프레임간 변화량을 이용한 가변 해상도 처리 기법을 제안한다. 4장에서는 프레임 속도를 증가하기 위한 대표적인 프레임 보간 기반의 다양한 연구들과 성능 그리고 한계성에 대해서 분석하고, 동일한 연산량을 기반으로 프레임 속도를 올리기 위한 해결 방안으로 타일 렌더링 기반의 GPU를 위한 프레임 속도 증가 기법을 제안한다. 5장에서는 데이터 재사용을 통한 GPU의 연산량을 줄이기 위한 최근 연구들을 분석하고, 최근 적용된 멀티 렌더 타깃을 활용한 데이터 재사용 기법을 통한 GPU의 메모리 및 연산량 최적화 기법을 제안한다.

6장에서는 첫째로 성능 평가를 위한 실험 환경 및 제안한 각각의 3가지 기법들의 효율성을 분석하기 위해서 널리 사용되는 대표적인 실험 벡터와 사용자 인지 능력 관점에서의 실험 평가 방법을 설명

한다. 최근 연구된 다른 기법들과 제안한 기법들을 사용자 인지 관점에서의 화질 평가, GPU의 연산량 그리고 전력 소모 관점에서 비교 분석한다. 7장에서는 본 논문의 연구 성과들의 요약 및 결론을 기술한다.

제 2 장

연구 배경

2.1 모바일 그래픽스의 발전

실시간 환경 이라는 관점에서 높은 프레임 속도에서 다양한 멀티미디어 콘텐츠를 처리하기 위해 모바일 하드웨어의 발전과 이를 최적화하기 위한 소프트웨어의 지속적인 연구와 발전이 지금의 스마트 폰과 태블릿 시장을 이끌어 왔다. 다음 2.1.1 과 2.1.2 절에서는 현재의 사용자의 요구사항에 따른 모바일 그래픽스 하드웨어 및 소프트웨어의 발전과 한계에 대해 설명할 것이다.

2.1.1 모바일 그래픽스 하드웨어의 진화

이번 절에서는 하드웨어 관점에서 모바일 GPU의 발전을 설명할 것이다. 모바일 GPU의 발전에서 두 가지 주요한 요인들로 설명이 가능하다. 해상도 증가와 다양하고 화려한 효과를 위한 GPU렌더링의 복잡도 증가에 의한 발전이다. 첫째로, 해상도 관점에서 요즘 상

용화되고 있는 스마트 폰은 5인치 화면크기에 최소 FHD의 해상도를 요구한다. 하지만, 앞서 언급했듯이, 최근 발표되는 스마트 폰은 QHD를 제공하고 있으며, 향후 UHD까지도 차별화 전략으로 고려하고 있다[66, 68]. 특히 이러한 해상도의 차이는 모바일 기기를 큰 화면의 TV에 연결했을 때 그 차이를 확연히 확인할 수 있다. 해상도가 커지면서 GPU의 연산량은 증가하고, 이로 인해 실시간 처리가 어려워지고, 이는 사용자에게 높은 응답속도를 보장하기 어려워진다.

둘째로, 보다 사실적이고 뛰어난 그래픽을 제공하기 위해 GPU가 렌더링 하는 복잡도에 따른 발전을 고려할 수 있다. 사실, 초기 휴대 전화는 특정 그래픽스 하드웨어가 포함되지 않고, CPU에서 단순한 소프트웨어 렌더링 같은 그래픽스 연산을 수행하였다. 그러나 위에서 언급했듯이 해상도의 증가와 보다 높은 수준의 그래픽 품질을 요구하게 되면서, 하드웨어 가속이 요구되었고, 모바일 GPU는 SoC의 핵심 구성 요소가 되고, 최근 몇 년 동안 급속한 발전을 이루었다. 초창기 모바일 GPU는 고정된 파이프라인으로 구현되어 OpenGL ES 1.1 API[70] 지원하였으며, 고정된 기능의 하드웨어로 프로그램이 불가한 구조(Non-programmable hardware) 로 100 Mega pixels/second 의 픽셀 처리 성능으로 시작했다. 이를 기반으로 모바일 GPU는 Desktop GPU 발전과 유사하게 진화하였다. 병렬처리의 기능 향상과 고정된 파이프라인 구조를 제거하고 쉐이더를 사용한 프로그램 가능한 하드웨어 구조(Programmable hardware)로 진화하였다. 현재 상용화되어 있는 스마트 폰은 멀티 코어 환경

에서 OpenGL ES 2.0 / 3.0 API[71]를 지원 하며, 최근 발표된 스마트 폰의 경우는 OpenGL ES 3.1 API[72] 까지 지원한다. 이를 위해 Giga pixel 단위의 픽셀 처리 성능을 제공해야 하며, 렌더링의 최소 단위인 삼각형을 초당 수백만 개 연산이 가능해야 한다. [69, 53, 77].

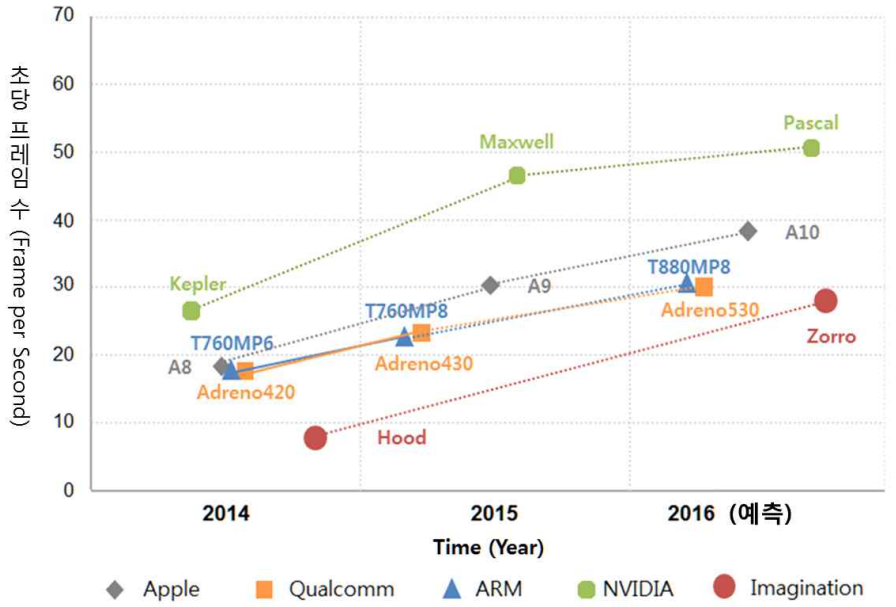


그림2.1 모바일 그래픽스 하드웨어의 발전

그림 2.1 에서는 현재 모바일 시장에서 선도하고 있는 주요 GPU IP 업체들의 지난 2년간의 GPU 하드웨어의 진화와 향후 1년의 성장을 예측 및 비교해 보았다. 비교를 위해 가장 최근 발표된 Kishonti사의 GFXBenchmark[83]의 GPU의 성능을 측정하기 위한 표준으로 사용되는 대표적인 테스트 벡터를 사용하였다.

Nvidia사의 경우, Tegra 시리즈의 SoC[Nvid5a]를 지속적으로 발표하였으며, Desktop용 GeForce GPU[57] 를 모바일에 적합하도록 저전력 GPU를 개발하여 왔다. Nvidia의 GPU의 경우는 Desktop GPU의 렌더링 방식을 그대로 가져와 렌더링을 한꺼번에 처리하는 직접 방식의 렌더링(Immediate-mode rendering)[1]을 채택하였으며, 다른 경쟁사 IP 대비하여 GPU 개발의 오랜 역사와 함께 높은 성능을 보여주나, 한꺼번에 처리해야 하는 방식으로 인해 메모리 대역폭 관점에서 취약하다.

Apple사의 모바일 GPU의 경우 Imagination사의 아키텍처[77] 를 채택하여, 해당 GPU의 하드웨어 구조와 소프트웨어의 최적화를 진행하여 Apple사의 이름으로 GPU를 발표하고 있다. Apple사의 경우, 비록 절대적인 성능 비교에서는 Nvidia사 보다 낮은 성능을 보여주고 있으나, 소모 전력 대비 효율 측면에서 가장 우수한 성능을 나타내며, 아이러니 하게 Imagination사의 GPU는 Apple사의 GPU 성능을 따라가지 못하고 있다.

Imagination사와 ARM 사의 경우, Power VR[77] 시리즈 와 Mali 시리즈 [69] 로 모바일 GPU를 연구 개발해 왔다. Desktop 방식에서 사용했던 오랜 전통의 직접 방식의 렌더링을 대신하여, Mobile SoC에 적합하도록 타일 기반 렌더링(Tile-based rendering) 방식[1]을 채택하였다. 렌더링 해야 하는 하나의 전체 프레임을 작은 타일 단위로 분할하여 GPU 내부의 메모리에서 처리하도록 하여 외부 메모리

의 접근을 최소화하여 메모리 효율성을 높이는 대표적인 방법이다.

마지막으로 Qualcom사의 경우, Adreno 시리즈로 모바일 GPU 를 발표하고 있다. Mali 시리즈와 유사한 성능을 보여주고 있으며, Adreno의 주요 특징은, 유연한 렌더링(Flex rendering) 기술[54] 을 지원한다. 렌더링 실행 시, 필요에 따라 직접 렌더링 방식과 타일 기반 렌더링 방식을 둘 다 지원하는 하이브리드 구조로 유연성 관점에서 강점을 가진다.

요즘 상용화되는 스마트 폰의 디스플레이 하드웨어는 기본적으로 1초당 60장의 프레임(60Hz)의 디스플레이를 지원한다. 앞서 설명했듯이, 높은 수준의 그래픽 품질과 사용자 환경을 제공하기 위해서는 모바일 GPU는 1초에 60장의 프레임을 렌더링을 해야 하는 성능을 보장해야 한다. 하지만, 그림 2.1 에서 볼 수 있듯이, 향후 1년 까지도 현 시점에서 존재하는 가장 복잡한 응용프로그램인 벤치마크의 성능 측정 결과가 60fps(Frame per second)를 만족하지 못한다. 더욱이, 최근 활발히 연구되고 있는 VR(Virtual reality)[42] 을 지원하기 위해서는 120fps를 만족해야 하지만, 이 요구사항의 절반도 미치지 못하는 것을 확인할 수 있다. 가장 치명적인 것은, 비록 성능을 만족할 수 있다 하더라도, 모바일 기기의 제한된 전력으로 이러한 성능을 지원하는 것이 근본적으로 불가능하다.

2.1.2 모바일 그래픽스 소프트웨어 진화

이번 절에서는 운영체제부터 하드웨어의 디바이스 드라이버까지 그래픽스 API를 포함한 모바일 그래픽스의 전반적인 소프트웨어의 발전에 대해서 설명하고자 한다. 현대의 모바일 기기의 진화는 이제 더 이상 전화와 문자 메시지만의 기능으로 제한되지 않는다. 스마트 기기로 발전되어, 웹 브라우징, 지도, 게임, 소셜 네트워크, 사진 및 비디오 편집과 같은 다양한 멀티미디어의 지원까지 그 범위가 매우 넓어졌다. 모바일 SDK[59, 61]는 이러한 요구 사항들을 만족하는 응용프로그램을 만들 수 있는 필요 라이브러리들과 함께 다양한 함수들을 제공하며, 특히 그래픽스 측면에서, OpenGL ES[62]를 통해 그래픽스 하드웨어 가속을 통해서 3D를 구현 및 재현을 지원한다.

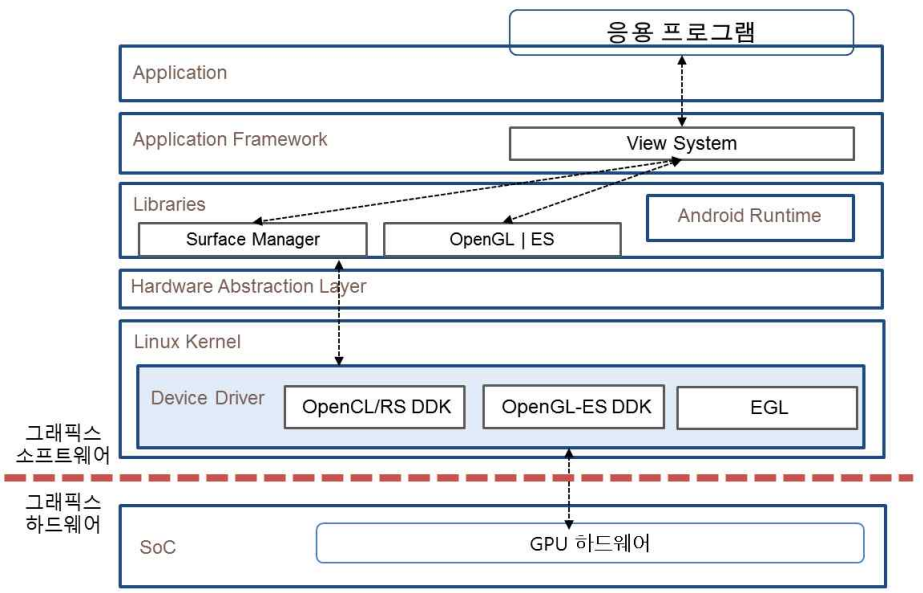


그림 2.2 모바일 그래픽스 관점에서의 소프트웨어 구조

운영체제 관점에서 보면, Google사의 Android와 Apple사의 iOS가 현재 스마트 폰 시장[13]을 지배하고 있다. 그래픽스 분야에서 두 운영체제는 2D / 3D 렌더링을 OpenGL ES 통해 다양한 그래픽스 콘텐츠와 높은 수준의 그래픽 사용자 인터페이스를 제공한다. 이를 위해 GUI 컴포지터(Compositor)는 필수적인 구성요소 이며, 일반적으로 동시에 실행하는 다수의 그래픽스 응용프로그램에 의해 생성된 다수의 그림을 하나의 그림으로 합성하는 프로세스이다. 빠른 사용자의 응답성을 보장하기 위해 이러한 합성 프로세스를 GPU가속기[55]를 통해서 처리 하거나 지정된 하드웨어 가속기[84]를 통해서 처리한다. 그림 2.2 에서 보듯이, 안드로이드 환경에서 그래픽스 응용프로그램을 수행하는 전반적인 흐름을 포현하고 있다. 그래픽스 응용프로그램에 의해 생성된 다수의 그림과 GUI 합성을 소프트웨어적으로 뷰 시스템을 통한 서피스플링거(Surfacefliger)[60]를 통해 처리한다. 응용 프로그램의 경우는 사용된 다양한 OpenGL ES API에 따라 렌더링 되지만, 합성의 경우에는 OpenGL ES 1.1 기반으로 처리할 수 있도록 정의되어 있으며, 모든 과정을 GPU를 통한 가속을 통해 처리한다.

내장형 시스템을 위한 OpenGL ES는 2D 와 3D 그래픽스 렌더링을 위한 OpenGL API의 일부로서, 데스크톱 GPU의 지원을 위한 OpenGL 기반으로 중복된 함수를 제거하고, 복잡한 기능을 단순화 함으로서 저전력 시스템을 위해 재정의 되었다. 지금까지 데스크톱의 OpenGL부터 다양한 API들이 Khornos 그룹[62] 의해 정의 및 표준화 되어 왔다. 앞서 언급되었듯이 많은 내장형 시스템 기반 많

은 개발 업체들이 모바일 그래픽스 분야에서 GPU를 활용하기 위해 OpenGL ES는 모바일 그래픽스에서 가장 널리 사용되는 지배적인 API 이다.

OpenGL ES 1.0과 1.1은 고정된 파이프라인의 GPU를 활용하기 위한 표준으로, 최근 접할 수 있는 고사양의 그래픽 품질을 재현하는 부분에서 많은 제약 사항이 있다. OpenGL ES 2.0으로 오면서, 고정 파이프라인이 아닌, 프로그램 가능한 파이프라인으로 정점(Vertex)과 프레그먼트(Fragment) 쉐이더를 통한 보다 복잡하고 다양한 렌더링이 가능했다. 예를 들어, 동적 그림자(Dynamic shadows)[4], 반사(Reflections)[5], 파티클 시스템(Particle systems)[7] 등 다양한 그래픽 효과들의 재현이 가능하게 되었다. 지금도 우리가 대부분 접하고 있는 많은 게임이나 멀티미디어 콘텐츠들은 아직도 OpenGL ES 2.0 기반이다. OpenGL ES 3.0을 지원하는 상용화된 모바일 기기가 나오기 시작한지는 얼마 되지 않았으며, 다양한 그래픽 효과들과 고 사양의 그래픽 품질을 재현하기 위한 추가적인 기능을 제공한다. 대표적인 예로, 폐색 쿼리(Occlusion queries)[58], 변환 피드백 (Transform feedback)[63], 그리고 멀티 렌더 타깃(Multiple render targets)[64]이 있다. 또한 추가적으로 메모리 대역폭을 줄이기 위한 추가적인 텍스처 압축 기술들 (Texture compression)[9], 예를 들어 ASTC[49]도 포함 되었으며, 프레임 버퍼 무효(Frame buffer invalidation) 기술도 제안 되었다. 최근 나온 기술의 복수 렌더 타깃 기반에서 GPU의 연산량을 줄이는 기법에서 대해서 5장 에서 자세히 다루도록 하겠다. 마지막으로 가장 최근 발

표된 OpenGL ES 3.1[62] 에서는 GPU를 단순 그래픽스 연산만이 아닌 범용 연산 (GPGPU) 으로 까지 확대하기 위한 컴퓨트 쉐이더 (Compute shader)[65] 지원을 포함한다. OpenGL ES 표준의 변화는 그래픽스 하드웨어의 발전과 그래픽 콘텐츠의 복잡도를 기반으로 진행되었으며, 그림 2.3 에서와 같이 OpenGL ES의 변화와 함께 하나의 픽셀을 처리하는데 필요한 cycle 을 보여준다.

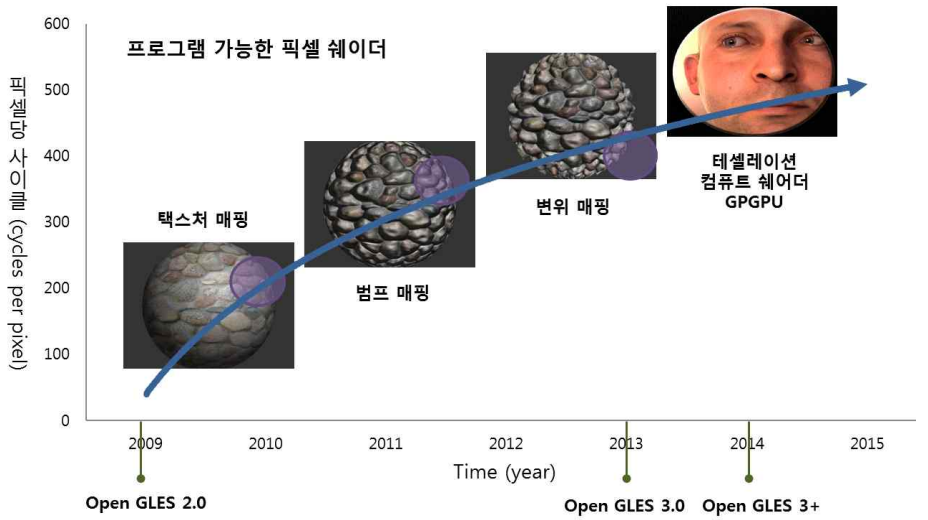


그림 2.3 모바일 그래픽 소프트웨어 발전과 쉐이더의 복잡도의 상관관계

최근의 소프트웨어의 사용자 요구사항을 만족하기 위해 하나의 픽셀을 처리 하는데 수백 cycle 이상의 연산을 필요로 한다. 이렇게 그래픽 소프트웨어의 진화에 따라, 연산의 복잡도가 증가하고, 모바일 GPU의 연산량이 증가한다. 결과적으로, 모바일 GPU의 급격한

소모 전력을 초래한다. 이로 인하여, 모바일의 제한된 전력 소모 관점에서 성능의 제한 및 발열 등 많은 문제가 야기된다. 지금까지도 저전력 모바일 환경을 위해 하드웨어적 저전력 설계 기술과 GPU 연산량을 줄이는 소프트웨어적 기술들이 많은 연구가 되고 있음에도 불구하고, 모바일 기기에서 작은 전력량으로 사용자 경험 측면에서 높은 수준의 그래픽 품질과 응답성을 보장하는 것은 쉬운 일이 아니다.

2.2 모바일 환경의 소모 전력 분석

앞에서 설명한 모바일 그래픽 하드웨어와 소프트웨어의 발전은 사용자에게 고품질의 그래픽 서비스를 제공함과 동시에 높은 응답성을 보장하는 것이 가장 쟁점이다. 그러나 점점 높아지는 고 사양의 그래픽 품질과 기능을 제공하기 위한 모바일 기기에서의 에너지 소모비용은 너무 크다. 즉, 제한된 전력으로 높은 응답성을 보장하면서 높은 그래픽 품질을 제공하기 위한 다양한 연구가 진행되어 오고 있음에도 불구하고, 아직까지도 해결하기 어려운 문제이다.

그림 2.4 에서는 가장 최근 상용화된 스마트 폰 LG G3 Screen [67]을 이용하여 모바일 기기의 그래픽스 성능 평가로 가장 많이 사용되는 벤치마크 중 가장 무거운 벡터인 Manhattan 3.0[83]을 실행했을 때의 전력 소모 분석 내용을 보여준다. 전력 소모의 주요 인자

인 디스플레이, CPU, GPU 그리고 메모리를 보여주며, 여기서 모뎀은 사용하지 않는다고 가정하였다. 화면의 해상도는 1920x1080을 지원하며 디스플레이는 60Hz로 동작한다. 즉 1초에 최대 60장까지 화면에 출력이 가능하다. CPU는 ARM CA15[73] 코어 4개와 CA7[73] 코어 4개로 각각 big.LITTE 클러스터 구조 [74]를 가지며, big 코어의 경우 최대 1.6 GHz까지, LITTLE 코어의 경우 최대 1.2 GHz까지 동작한다. GPU의 경우 Imagination사의 Rogue [78] 가 적용되어 있으며, 최대 450 MHz까지 동작 가능하다.

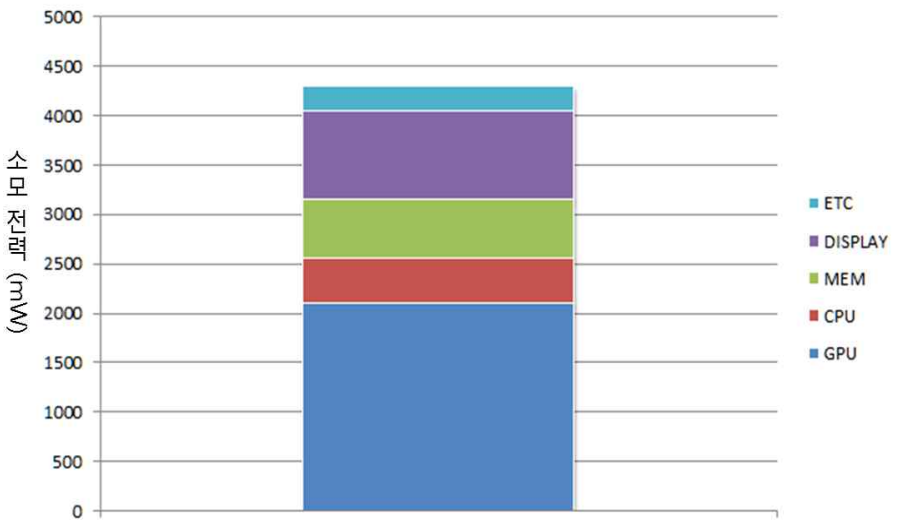


그림 2.4 LG G3 Screen 에서의 GFXBenchmark 의 Manhattan 3.0 의 AP 의 소모 전력 분석

GPU가 최대 성능으로 동작하고 있음에도 불구하고, 실제 그래픽 성능이 10 fps로 실제 디스플레이 디바이스의 최소 요구 조건인 60 fps를 만족하지 못한다. 그럼에도 불구하고, 그림 1.4에서 보여주듯

이 GPU가 모바일 기기가 소모하는 전체 전력 소모의 48%가 되는 평균 2100 mW 소모한다.

SoC설계 기술과 다양한 에너지 감소 기술이 연구되어 왔음에도 불구하고, 모바일 기기에서 작은 전력량으로 고정된 높은 해상도와 높은 프레임 속도를 달성하는 것은 쉬운 일이 아니다. 또한 전력 소모가 크다는 것은 곧 발생할 수 있는 발열이 크다는 것을 의미 하며, 발열이 클수록 동일한 조건에서 전력 소모는 증가한다. 그렇기 때문에 현대의 모바일 기기에서는 동일한 성능 조건에서 GPU의 하드웨어와 소프트웨어의 설계에 있어서 소모 전력 절감을 고려한 설계와 소프트웨어 적으로 GPU의 연산량을 줄이는 것이 핵심이라고 할 수 있다.

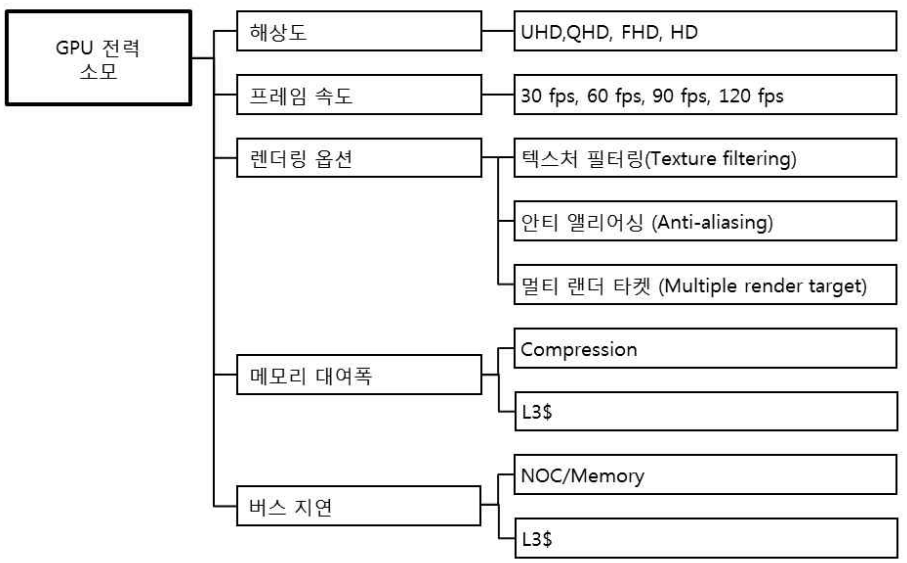


그림 2.5 GPU 전력 소모관점에서 주요 가변 요인

스마트 폰과 태블릿은 배터리에 의해 구동이 되기 때문에, 지원 가능한 렌더링에 있어서 가능한 적은 에너지를 소모할 필요가 있다. 사용자가 매 시간 마다 배터리를 충전을 하면서 사용하지 않기 때문에 한 번의 배터리 충전으로 사용가능 한 시간은 모바일 기기의 성능을 평가하는 데 매우 중요한 지표로 사용된다. 모바일 기기의 발전과 함께 모바일 GPU의 필요 소모 전력 증가 속도는 모바일 배터리 기술의 진화보다 더 빠른 속도로 증가하며[11], GPU의 하드웨어 성능은 사용자의 높은 요구 사항인 높은 해상도에서 고정된 최소한의 프레임 속도를 따라가지 못하고 있다. 따라서 배터리 충전 당 그래픽 처리를 오래 하기 위해서는 사용자 경험 측면에서 성능의 감소를 느끼지 못하는 수준에서 최적화하는 방법이 연구되어야 한다.

표 2.1 GPU 전력 소모 관점에서 최적화 가능 인자들

특성	조절 범위	기존 전력 소모 (mW)	전력 소모 최소 값 (mW)	최대 GPU 전력 감소량
Anti-aliasing 사용 유무/정도	4xAA → 2xAA → AA off	2100	1827	4.1 % (AA off)
Texture filtering 정도	Mipmap → Bilinear → Nearest	2100	1808	5.1 % (Nearest)
Precision of depth render buffer	32bit → 24bit → 16bit	2100	1844	3.2 % (16bit)
Multiple render target 재사용	재사용 RT 1장 → 2장 → 3장 → 4장	2100	1697	10.9 % (4장)
Screen resolution	1080P → 900P → 720P	2100	1421	25.4% (720P)
Frame rate	60fps → 45fps → 30fps	2100	953	50.1% (30fps)

이를 위해 그림 2.5 는 소프트웨어적 관점으로 응용 프로그램에서 하드웨어 제어를 위한 디바이스 드라이버까지 전력 감소를 위한 제어가 가능한 인자들을 보여준다. 모바일 GPU의 전력 감소에 있어서 가장 효과적인 인자들을 찾기 위해 인자들의 변경을 통해 반복 실

험을 진행하였다. LG G3 Screen[67] 개발보드 기반으로 20개의 샘플 칩을 가지고 GPU와 개발 보드 전체의 전력 소모를 측정하기 위해 DAQ NI USB-6363[79] 장비를 연결하여, GFXBenchmark의 Manhattan 실험 벡터[83]를 10번 반복 시행하였다. 시행 마다 하드웨어 초기화는 냉각기로 시행하여 발열로 인한 오차를 최소화 하였다. 보다 자세한 실험 환경은 6장에서 다룰 예정이다. 치명인자를 확인하기 위해 제어 가능한 인자의 변경을 통해 매 주기마다 수집된 전력 소모 결과에 대한 평균을 계산하였으며, 표 2.1 과 같이 GPU 전력 소모 관점에서 최적화가 가능한 치명 인자들을 확인이 가능하다.

2.3 해상도

앞서 언급 되듯이 요즘의 스마트 폰은 FHD(1920x1080) 이상의 높은 해상도를 제공하는 방향으로 발전하고 있다. 제조업체에서 발표하는 최근 모바일 기기는 QHD(2560x1400)를 시작으로 WXGA (3200x1800) 그리고 UHD(3840x2160)의 해상도로 발전해 가고 있다. 이렇게 해상도가 증가함에 따라 모바일 GPU의 연산량도 급격하게 비례하여 증가하게 된다. 모바일 GPU의 전력 소모는 GPU의 연산량에 거의 정비례로 증가하기 때문에, 해상도의 증가는 곧 GPU 전력 소모에 가장 직접 적인 영향을 끼친다. 이러한 전력 소모는 발열과 배터리 수명에도 당연히 직접적인 영향을 끼친다.

보다 선명한 그래픽 품질을 제공하기 위한 높은 해상도는 많은 시스템 자원을 요구하게 된다. 최근의 그래픽스 시스템은 렌더링을 위한 지연을 최소화하기 위해 다중 프레임 버퍼 구조를 적용한다. 보통 2개 혹은 3개의 프레임 버퍼를 외부 메모리에서 관리하게 된다. 일차적으로 GPU가 UHD 의 그림을 렌더링 한다고 가정하면, FHD로 필요로 하는 1920x1080 픽셀의 4배의 수만큼 렌더링을 해야 한다. 또한, 하나의 픽셀이 32bit 으로 표현된다고 하면, UHD 의 경우 그림 당 100MB 이상의 여러 장 그림을 외부 메모리에 쓰기 위해서는 FHD 의 4배의 메모리 대역폭이 필요로 하게 된다.

높은 품질의 렌더링에 대한 사용자의 요구와 저전력을 위한 에너지 효율에 대한 요구는 사실 상반되는 부분이다. 이러한 트레이드 오프는 사용자의 경험 입장에서, 인간의 눈이 인지할 수 없는 수준 이상의 품질을 배터리 수명에 대한 손해를 감수하면서 유지하는 것은 불필요하다. 반대로, 배터리 수명만을 위해서 높은 수준의 그래픽 품질을 포기하는 것도 해답은 아니다. 본 논문의 3장을 통해서 동적 해상도 기반 GPU의 연산량을 줄이는 최근 연구 방법들과 문제점들에 대해 분석한다. 그리고 이러한 문제점들을 해결하고, 그래픽 품질의 차이를 인지할 수 없는 수준을 유지하면서 모바일 기기에 적합하도록 GPU의 연산량을 줄이는 새로운 방법을 자세히 제안한다.

2.4 프레임 속도

고정된 높은 프레임 속도를 만족하는 것은 사용자의 응답성과 실시간 그래픽 품질 제공하는 점에서 매우 중요한 지표이다. 사용자의 그래픽 품질에 대한 요구 사항이 높아짐에 따라, PC 수준의 사실적인 그래픽 품질을 위한 복잡한 렌더링을 하는 응용 프로그램이 많아지는 추세이다. 더욱이 요즘 발표되는 모바일 기기는 대부분 60Hz 기반으로 1초에 최대 60장까지 화면에 디스플레이 하도록 되어 있으나, 위와 같은 사실적인 복잡한 렌더링을 하는 응용프로그램의 경우 GPU 하드웨어 렌더링 속도가 60fps의 디스플레이 기준에 미치지 못한다.

또한 그림 2.1 에서 확인 할 수 있듯이 최근 VR[42] 과 같은 새로운 기술들이 소개되면서 60Hz 기반의 디스플레이는 최대 120Hz까지 요구 되지만, 최근까지 발표된 상용화된 모바일 기기의 성능은 절반에도 미치지 못한다. 낮은 프레임 속도로 렌더링하는 경우, 사용자는 빠르게 움직이는 사물이나 급격한 카메라의 시점 변화에 의해 끊김 현상을 쉽게 확인 할 수 있으며, 높은 응답 속도도 보장할 수 없다. 이와 같은 사람이 쉽게 인식할 수 있는 끊김 현상을 개선하기 위해서 프레임 속도를 올리기 위한 다양한 연구들이 최근까지 이루어 졌다. 가장 널리 사용되는 기법으로 프레임 보간(Frame interpolation)으로 낮은 프레임 속도의 연속된 2장의 그림을 통해 중간의 그림을 생성하여 삽입하는 기법으로 가장 대표적인 방법이 정방향 재 투영 기법(Forward re-projection)과 역방향 재 투영

(backward re-projection)을 이용한 프레임 보간 기법이 있다.

하지만, 이러한 기법들을 통해 생성하는 중간 프레임은 높은 에너지 소모를 요구하며, CPU와 메모리 연산의 관점에서도 상당한 추가 비용이 든다. 비록 낮은 프레임 속도를 높은 프레임 속도로 증가 가능 하지만, 중간 프레임 생성을 위한 상당한 추가 비용 때문에 제한된 모바일 전력에서 구현하는데 한계가 있다. 최근 연구된 프레임 보간 기법의 한계점을 4장에서 보다 자세히 분석하고, 이러한 문제점을 해결하기 위하여 저전력 모바일의 환경의 타일 기반 렌더링 GPU를 활용한 새로운 프레임 속도 증가 기법을 제안하도록 하겠다.

2.5 데이터 중복

렌더링 프레임간의 상관관계인 시간적 일관성은 실제 실시간 렌더링에서 광범위하게 존재한다. 시간적 일관성을 유용하게 활용하면, 불필요한 계산을 감소시킬 수 있으며 또한 최소한의 품질 저하로도 데이터 재사용을 통해 렌더링 작업을 현저하게 줄일 수 있다. 이점을 잘 활용하면, 더욱 복잡한 계산을 필요로 하는 쉐이딩 연산을 비록 성능이 부족한 하드웨어 사양으로도 고품질의 그래픽 응용 프로그램 렌더링 구현이 가능하다.

이러한 재사용 기술은 가시화 컬링(Visibility culling) 최적화, 개체-공간 전역 조명 예측을 포함한 화소 재사용뿐만 아니라 멀티 패

스 음영 효과, 쉐이더 안티 엘리어싱(Shader anti-aliasing), 그림자 만들기 그리고 전체 조명 효과(Global illumination effects)들과 같은 다양한 쉐이딩 연산에서도 시간적 일관성을 통한 데이터 재사용이 활용 가능하다. 하지만, 최신 모바일 GPU는 OpenGL ES 3.x 이상을 지원하며, OpenGL ES 3.x를 활용하는 고품질의 그래픽 콘텐츠의 보급도 증가하고 있음에도 불구하고, OpenGL ES 3.x의 대표 기능 중 하나인 멀티 렌더 타깃(Multi-render target: MRT)에 적용하는 방법에 대해서는 아직 연구 성과가 없다.

멀티 렌더 타깃은 G-Buffer라 불리는 프레임 버퍼 객체(Framebuffer object: FBO)에 복수의 렌더 타깃을 두고 각 타깃들이 기하 데이터를 공유하여 한 번에 렌더링 한다. 멀티 렌더 타깃은 대표적으로 지연 쉐이딩(Deferred shading)에 활용되어 전통 적인 방식인 포워드 쉐이딩(Forward shading)과 달리 객체별로 광원 연산하지 않고 전체 장면을 한 번의 광원 연산으로 동일한 효과를 보여줄 수 있기 때문에 연산 효율을 증가시킬 수 있다.

하지만 동시에 하나 이상의 여러 개의 타깃에 렌더링 해야 하기 때문에 많은 메모리 대역폭을 필요로 하는 단점이 있다. 곧 높은 메모리 사용은 높은 전력 소모를 의미한다. 즉, 메모리 대역폭과 전력 소모에 치명적인 모바일 기기 환경에서는 큰 장애 요인일 수밖에 없다. 따라서 하드웨어 종속성이 없으며, 연산 비용이 적은 단순하면서 메모리 효율적인 데이터 재사용을 통한 최적화 기술이 필요하다. 데이터 재사용을 통한 최근까지 활용된 기술을 5장에서 살펴보

고, 적용된 기본 개념을 이용하여, 모바일의 저전력 환경에서 적합하도록 멀티 렌더 타깃을 위한 데이터 재사용 기법에 대해서 제안한다.

제 3 장

가변 해상도 기반 최적화

3.1 거리 기반 가변 해상도 변환 기법

최근 발표되는 고 해상도의 모바일 기기에서 고정된 높은 해상도는 GPU 렌더링 관점에서 추가되는 픽셀들로 인하여 많은 연산량과 메모리 대역폭을 필요로 한다. 곧 이러한 요인들은 전력 소모에서 가장 치명적인 부분이며, 그림 3.1에서 GPU의 연산량과 전력 소모는 해상도와 거의 정비례 하는 상관관계를 보여준다. 최근 이를 개선하기 위한 다양한 연구가 진행되고 있으며 고정 해상도가 아닌 가변 해상도 기반으로 해상도 조절에 있어서 무엇을 기준으로 할지에 대한 다양한 연구가 이루어지고 있다.

사람의 인지 능력의 기준을 사람의 눈과 모바일 기기간의 거리를 기반으로 해상도를 조절[21]하는 연구가 마이크로소프트 연구 센터에서 제안 되었으며, 디스플레이의 밀도와 사람 눈과의 거리의 상관관계를 하기와 같이 정의하였다.

$$N = \frac{L}{2D\ tan\left(\frac{\delta}{2}\right)}$$

여기서 N 는 픽셀 수를 의미하며, L 은 디스플레이의 긴 쪽의 길이를 나타낸다. D 는 사용자와 디스플레이 화면과의 거리를 이며, δ 는 사용자의 시점과 디스플레이 화면간의 각을 의미한다. 위의 정의를 기반으로 현재사용자와 디스플레이간의 거리에 따라 사용자 인지 능력 관점에서 최적의 픽셀 수가 정해지고, 정해진 픽셀 수에 대응되는 해상도가 결정된다.

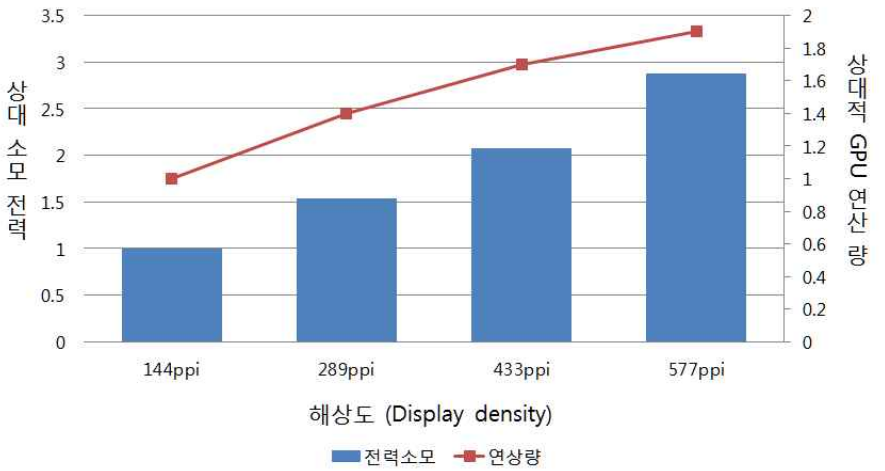


그림 3.1 디스플레이 해상도와 GPU 연산량 및 소모 전력의 상관관계

결정된 해상도를 현재 GPU 렌더링 과정에 반영하기 위해서,

OpenGL ES 관점에서 처리한다. 안드로이드 환경에서의 GPU 렌더링은 최소한 다른 2개의 렌더 타깃인 사용자 정의 렌더 타깃(User-defined render target)과 디폴트 렌더 타깃(Default render target) 을 사용한다. 이와 같은 하나 이상의 렌더 타깃은 동적 범위(High-dynamic range)[22], 그림자[23] 그리고 필드 깊이 연산(Depth of field)[24] 등 다양한 그래픽 알고리즘들을 처리하기 위해 활용된다. GPU 연산량 감소를 위해서는 두 렌더 타깃을 함께 조절해야 효과적이므로, 이를 위해 API 호출을 중간에 읽어 처리하도록 하였다. 결과적으로 사람 눈과 거리가 멀어질수록 높은 해상도에서 낮은 해상도로 렌더링 하더라도, 사람의 인지 능력 관점에서 그 차이를 크게 느끼지 못한다는 것이다.

디스플레이와 사용자 눈과의 거리를 기반으로 한 사용자 인지 능력을 활용한 접근 방식은 상당히 유효한 결과를 보여주었다. 하지만, 치명적인 몇 가지 문제가 있다. 사용자의 시력은 동일하지 않으며, 사용자의 눈을 인식하기 위해 센서기반으로 눈과 거리를 인식한다. 일반적인 상황에서는 센서로부터 유효한 값을 얻어 올 수 있겠지만, 예를 들어 자연광, 안경 그리고 온도 등 외부적 환경 요인으로 언제나 유효한 값을 읽어 온다는 보장을 할 수 없다. 결과적으로 잘못된 값은 잘못된 그래픽 품질을 야기하게 된다.

3.2 응용 프로그램 특성 기반 해상도 변환 기법

최근 또 다른 접근 방식으로 Qualcom사는 응용 프로그램 기반 해상도 조절 기법을 발표[53] 하였다. 모바일 기기에 적용하기 위해서, 미리 광범위한 응용 프로그램들의 특성을 분석한 데이터베이스를 구축하여, 모바일 기기에서 이미 분석된 데이터베이스로 부터 응용 프로그램의 특성에 따라 실행하는 응용 프로그램 특성 기반의 해상도를 조절하는 방법이다. 기본적으로 특정 응용 프로그램에서 한번 정해진 해상도는 응용 프로그램을 종료하기 전까지 환경에 따라 적응적으로 변하지 않고 그대로 유지된다. 유사한 방법을 따르고 있는 Lucid사의 PowerXtend[25]도 응용 프로그램의 특성에 따라 해상도를 조절한다. 특성 이외 추가적으로 사용자의 상호 작용이 클 경우, 현재 GPU의 렌더링이 기대 성능에 만족하지 못하면, 높은 사용자의 응답속도를 보장하기 위해 낮은 해상도로 렌더링 한다.

위의 최근 연구된 모든 접근 방법은 기본적으로 미리 분석된 응용 프로그램의 특성을 기반으로 하기 때문에, 응용 프로그램의 특성이 바뀌게 되면 유연성 있는 대응이 불가능하다. 추가적으로 지정된 응용 프로그램의 환경에 사용자의 다양한 시나리오를 모두 대응하기에는 한계가 있다. 그러므로 언제나 예상하는 그래픽 품질을 보장할 수 없으며, 적응적이지 못한 접근 방식으로 그래픽 품질의 많은 결점을 야기할 수 있다.

이번 절에서 언급한 문제들의 근본적인 해결을 위해 그래픽 콘텐츠를 고려한 다양한 시나리오에 적응적으로 대응할 수 있는 연구가 필요하다. 위의 문제를 해결하기 위해 사용자의 인지 능력을 기반으로 하여, 모델 뷰 투영 행렬을 이용한 프레임 간 변화량 기반으로 한 가변 해상도 조절 기법[3, 6]을 다음 절을 통해 제안한다.

3.3 동적 렌더링 기반 전력 소모 최적화 및 품질 개선

최근 연구된 해상도 기반의 접근 방식들은 앞서 언급했듯이, 치명적인 문제점을 가지고 있다. 응용 프로그램의 특성 기반 접근 방식은 미리 분석된 응용 프로그램의 특성만을 기반으로 하기 때문에, 응용 프로그램의 특성이 바뀌면 유연한 대응이 불가능하다. 또한, 응용 프로그램의 특성만을 고려했을 뿐, 실제 사용자의 인지적 능력은 고려하지 않았기 때문에 사용자 관점에서 최적화는 전혀 고려되지 못했다. 반면에 사람의 인지 능력을 고려한 접근 방법 중 디스플레이와 사용자 눈과의 거리를 활용한 방식은 일반적인 상황에서는 유효한 결과를 보였다. 하지만, 사용자의 시력은 동일하지 않으며, 해상도 조절은 센서를 통해 획득한 데이터를 이용하기 때문에 센서는 언제나 신뢰할 수 있는 값을 보장하여야 한다. 하지만, 자연광, 안경, 온도 등 외부적 환경 요인으로 언제나 유효한 센서 값을 보장하기 어렵다.

이러한 문제점들을 해결하기 위해 사람의 인지 능력을 고려하여 그래픽 콘텐츠 기반인 프레임간의 변화량 분석을 통한 가변 해상도 기법을 제안[3, 6] 한다. 해상도 조절에 있어서, 핵심 개념은 사람의 인지 능력관점에서 프레임 간 빠르게 움직이는 객체가 많을 경우 일정 속도 이상의 프레임 속도에서는 뭉개져 보이는 점[26]을 활용한다. 이 특성을 활용하여, 프레임 간 변화량이 많을 경우, 즉 프레임 간 많은 객체의 큰 변화가 있을 경우, 낮은 해상도로 렌더링하여, 사람의 눈에 뭉개져 보이는 효과의 유사한 결과를 통해 높은 해상도로 인한 렌더링 비용을 최적화 한다. 하지만 낮은 해상도로 렌더링하여 높은 해상도로 스케일링 할 경우, 사람이 인지 가능 할 정도의 뭉개짐 효과가 그래픽 결점으로 발생할 수 있다. 이러한 결점을 개선하기 위해 낮은 해상도 프레임을 렌더링 할 때, GPU의 다양한 렌더링 옵션을 조절하여, 낮은 해상도 프레임의 경계와 전반적인 그래픽 품질을 보다 선명하게 렌더링하여, 뭉개짐 현상을 최소화 한다. 결과적으로 이렇게 선명하게 렌더링 된 낮은 해상도 프레임은 스케일링 시 발생하는 뭉개짐 효과를 어느 정도 보상하여 원본 수준의 그래픽 품질로 복원이 가능하다. 그림 3.2 에서는 제안하는 프레임 변화량 기반 가변 해상도 기법의 시스템 전반적인 흐름을 보여준다.

간략하게 시스템 전반적 흐름을 보면, 제안하는 기법은 소프트웨어 계층에서 그래픽 라이브러리와 디바이스 드라이버 사이에 위치한다. 먼저, 응용프로그램의 API 호출을 가로채어 프레임 간 변화량을 분석하기 위하여 모델 뷰 프로젝션 행렬(Model view projection

matrix: MVP)의 값들을 얻는다. 프레임 마다 읽어온 행렬들의 변화 정도를 기반으로 프레임 간 변화량을 분석한다. 분석된 프레임간의 변화량을 기반으로 렌더링 할 프레임의 해상도를 조절하고, 높은 해상도와 낮은 해상도의 비율을 결정한다. 이렇게 결정된 비율과 해상도를 통해 높은 해상도와 낮은 해상도를 번갈아 가면서 렌더링하며, 추가 적으로 GPU의 렌더링 옵션을 통해 낮은 해상도 프레임의 스케일링으로 인한 화질 열화를 개선한다.

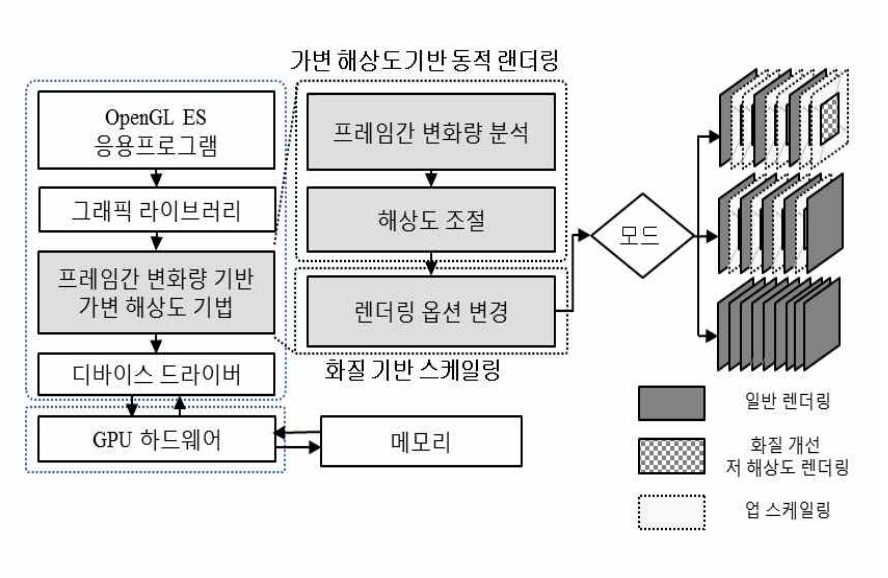


그림 3.2 프레임 간 변화량 기반 가변 해상도 시스템

3.3.1 인간 시각 시스템 기반 동적 렌더링

제안하는 해상도 기반 동적 렌더링은 고정 해상도가 아닌 가변

해상도를 통해 프레임마다 다른 해상도로 렌더링을 한다. 가변 해상도 기법을 통해 고정 해상도로 인한 높은 GPU의 연산량을 감소 시켜 결과적으로 GPU의 소모 전력을 줄일 수 있다. 다른 측면으로, 상대적으로 낮은 GPU의 하드웨어 성능으로도, GPU의 높은 연산량을 요구하는 응용프로그램의 렌더링 속도 fps를 자연스럽게 올리는 결과를 가져올 수 있다. 사용자 관점에서 보다 높은 그래픽 품질의 서비스를 받을 수 있으며, 보다 높은 사용자 응답 속도를 보장할 수 있다.

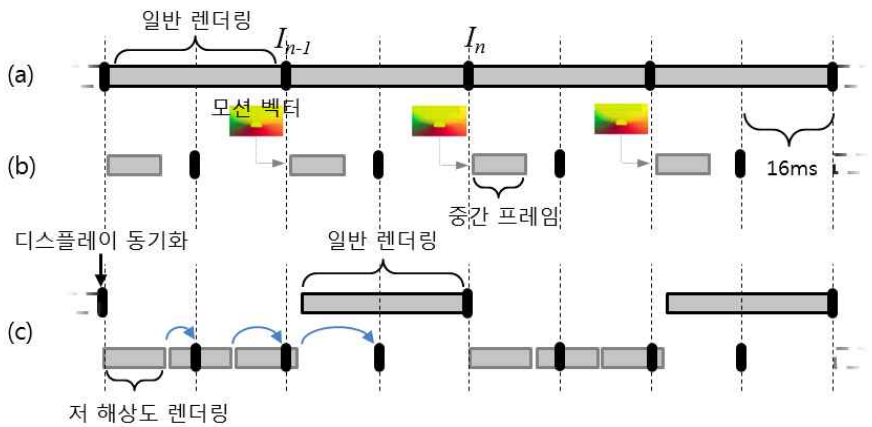


그림 3.3 가변 해상도 기법을 통한 동일한 GPU 연산량으로 프레임 속도 증가 시나리오

(a) GPU가 30fps로 렌더링, (b) 일반적인 프레임 보간 기법,

(c) 제안하는 가변 해상도 기법

시나리오 관점에서 그림 3.3 (a)와 같이 디스플레이의 동기화 신호 속도는 (1/60초 = 16ms)을 지원하지만, GPU 하드웨어 성능의 부족으로 주어진 응용 프로그램을 30fps 성능으로 렌더링하는 경우를 보여준다. 이 경우, 초당 60장의 속도를 만족하기 위해서는 가장 일반적인 방법인 프레임 보간 기법에 의하여 중간 프레임 생성하여 60장의 성능을 만족 수 있으며, 그림 3.3 (b)와 같다. 하지만, 모션 벡터 기반의 재 투영 방식의 프레임 보간 기법은 중간 프레임 생성을 위한 높은 연산 비용으로 모바일의 저전력 환경에서는 적용이 불가능 하다. 이러한 문제를 해결하기 위해 그림 3.3 (c)와 같이 제안하는 가변 해상도를 통한 동적 렌더링 방법은 매우 적은 추가 비용으로 기대 성능까지 프레임 속도를 올릴 수 있다.

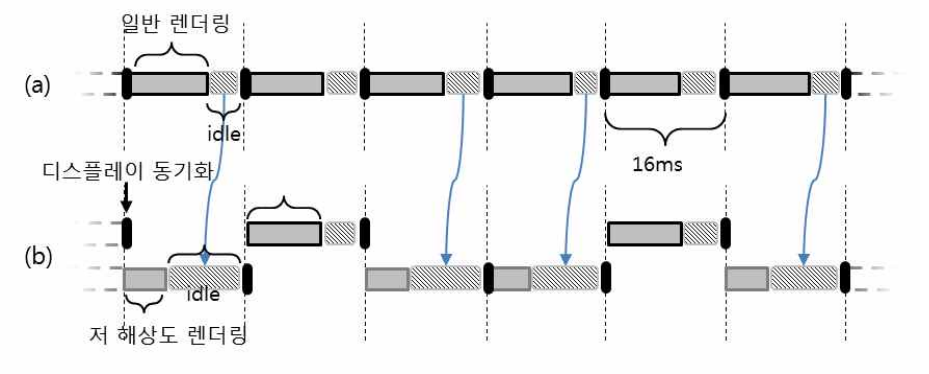


그림 3.4 가변 해상도 기법을 통한 동일한 성능 기준 GPU 연산량 감소 시나리오

(a) GPU가 60fps로 렌더링, (c) 제안하는 가변 해상도 기법

반면에 그림 3.4 (a)는 GPU의 하드웨어 성능이 1초의 60장 이상의 렌더링 속도를 보장하는 경우를 보여준다. GPU 성능이 충분히 높기 때문에, 주어진 1/60 초 동안에 렌더링을 완료하면, GPU는 에너지 절감을 위해 idle 상태로 들어간다. 그림 3.4 (b)는 제안하는 가변 해상도를 이용한 동적 렌더링 기법을 통해 동일한 고정 해상도로 렌더링하는 대신에, 높은 고정 해상도 사이에 낮은 해상도의 렌더링을 통해 idle 시간의 증가를 보여준다. 결과적으로 이렇게 얻어진 idle 시간의 증가는 곧 GPU의 전력 감소를 의미한다.

3.3.2 변환 행렬을 통한 변화량 계산

높은 해상도 프레임 렌더링 사이의 낮은 해상도 프레임 렌더링 비율을 정의는 계산된 프레임 간 변화량을 기반으로 한다. 프레임 간의 변화량을 계산하기 위해서, 모델 뷰 프로젝션 행렬을 이용한다. 먼저, 모델 뷰 프로젝션 행렬은 GPU에서의 프레임을 구성하는 객체들의 변환 프로세스를 표현하는 행렬이다. 각각의 진행 과정은 다음과 같다. 모델링 변환 단계, 뷰 변환 단계, 프로젝션 변환 단계, 뷰 포트 변환 단계를 진행한다. 여기서, 모델 행렬(Model matrix)의 모델은 정점들의 집합으로 정의된다. 정점들의 x, y, z 좌표계는 객체의 중심을 고려하여 정의된다. 즉, 만약 정점이 (0,0,0)에 있다면, 이 좌표계는 객체의 중심에 있게 된다. 모든 정점들은 모델의 중심을 기준으로 정의되어 있는 모델 공간(Model space)에서 모든 정점들이 월드 공간(World space) 기준으로 옮기기 위한 행렬을 의미한

다.

행렬 연산은 3D 공간에서 정점으로 폴리곤(Polygon)을 생성할 수 있으며, 이러한 폴리곤 들을 통해 3D 공간에 객체처럼 그려지게 된다. 이러한 폴리곤들은 GPU에서는 쉐이더에서 정점 처리(Vertex processing)를 통해 하기에서 정의한 행렬로 실제 디스플레이에 보일 위치로 변환하는 과정이 필요하다. 모델링 변환 단계(Modeling transformation, Model-view matrix)는 모델 공간을 실제 공간으로 변환하는 것을 말한다. 여기서, 정점(x, y, z)은 원점을 기준으로 표현된 객체의 좌표점이다. 뷰 변환 단계(View transformation)는 실제 공간을 카메라 공간으로 변환하는 것을 말한다. 뷰 변환 단계는 모델을 만들기 위해 사용한 기본 모델(raw model) 좌표계를 관측점(viewpoint)에서 보았을 때 위치(눈을 원점으로 간주)하는 좌표계로 변환한다. 이때 사용되는 행렬은 모델-뷰 행렬(Model-view matrix)이다. 즉, 첫째 변환에서는 원점을 기준으로 하는 객체 좌표계(기본 모델 좌표계)의 좌표 점을 관측점인 눈 혹은 카메라를 원점으로 하는 실세계 좌표계의 한 점으로 변환한다. 뷰 행렬은, 월드 공간(World space)으로 옮겨진 모델은 다시 카메라 공간으로 이동해야 하는데, 카메라 공간에서 모든 정점들은 카메라를 기준으로 정의되며, 해당 변환을 위한 행렬을 말한다.

프로젝션 변환(Projection transformation, Projection matrix) 단계는 관측 볼륨에서(viewing volume)에서 벗어난 영역을 제거한다. 따라서 화면에서 볼 수 있는 물체만을 가지게 된다. 관측 볼륨은 관측

자 시야에 들어오는 영역을 의미한다. 투영 변환에 의해서 정점이 앞에 있는지 뒤에 있는지 3차원에서의 위치가 정해지고 정점들을 정규 화된 장치 좌표(Normalized device coordinate)로 변환을 한다.

알고리즘:
현재 프레임 F_i에 대해 동적 렌더링을 위한 파라미터 P_i계산
Input parameter: 현재 프레임 F_i, GPU 렌더링 상태 정보 와 링크드 리스트 L_{i-1}
Return: 높은 해상도와 낮은 해상도의 렌더링 비율 P_i

```
1:   function CalcDynamicRenderingParam(F_i, L_{i-1})
2:     Initialize Linked list L_i
3:     while ( j < Total count J of Objects in F_i)
4:       Read the j^th Object's ID; O_ij
5:       Read corresponding MVP matrix; M_ij
6:       Compute V_ij=M_ij * I
7:       Search V_{(i-1)j} in the linked list L_{i-1} using O_ij
8:       Compute D_ij = distance(V_ij, V_{(i-1)j})
9:       Insert the node(O_ij, V_ij, D_ij) to the linked list L_i
10:    Search (D_max) in the linked list L_i
11:    Normalize D_max to D_nor
12:    Calculate the average of D_nor
13:    Select P_i from comparing the avg. of D_nor with pre-defined threshold
14:    L_{i-1} = L_i; discard the i-1^th linked list
15:  return P_i
```

그림 3.5 동적 렌더링을 위한 높은 해상도와 낮은 해상도의 렌더링 비율 계산 알고리즘

프로젝션 매트릭스는, 화면에 객체를 어디에 놓을지 결정할 때 x 와 y 좌표뿐만 아니라 z 값도 사용해야 한다. 더 큰 z 값을 가지는 정점은 다른 정점들 보다 화면의 앞으로 오게 되듯이, 화면에 객체를 투영하기 위한 매트릭스를 말한다. 뷰포트 변환(Viewport

transformation)은 3차원 물체를 보여줄 윈도의 좌표(2D 좌표계)로 변환한다. 위에서 설명한 일련의 변환 단계들은 OpenGL 혹은 OpenGL ES 에서는 3차원 상의 정점을 2차원 평면에 나타내기 위해 4x4 행렬로 표현이 가능하며, 정점의 좌표에 이들 행렬을 곱함으로써 변환을 하게 된다.

결과적으로 모델 뷰 프로젝션 행렬(MVP)은 모델 행렬, 뷰 행렬 그리고 프로젝션 행렬 모두 합쳐서 생성된다. 즉, 이 행렬의 변화를 통해 프레임을 구성하는 객체들의 변화 정도를 계산할 수 있다. 프레임을 구성하는 객체들의 변환을 표현하는 이 행렬들을 얻기 위해서는 먼저 *gl call* 단위에서의 API 호출을 가로채는 것으로 시작한다. 그림 3.5 는 동적 렌더링을 위한 파라미터 P_i, 즉 높은 해상도의 프레임 사이에서 낮은 해상도의 렌더링 비율을 구하는 알고리즘의 주요 단계를 보여준다.

여기서, F_i 는 렌더링 순서에서 i 번째 프레임을 나타내며, j 는 하나의 F_i 프레임을 구성하는 객체들의 총 개수를 의미한다. 일반적으로 하나의 프레임은 여러 개의 객체들과 각 객체들에 상응 하는 MVP 행렬로 이루어져 있기 때문에 *Object ID*는 한 프레임 내에 객체들을 구별하기 위해 사용한다. 결과적으로 O_{ij}는 i번째 프레임의 j 번째 *Object ID* 이다. 마찬가지로, M_{ij} 는 i번째 프레임의 j 번째 MVP를 나타내며, 상응하는 O_{ij} 와 하나의 쌍을 이룬다. V_{ij} = (x, y, z, w=1) 는 MVP 행렬에 I=(1, 1, 1, w=1) 의 행렬변환을 통해 얻어진 값으로, I=(1, 1, 1, w=1)의 MVP 행렬 변환 목적은 MVP의 행렬

간의 변화량을 정도 차이를 계산하기 위해 고정된 값을 정의하여, 결과적으로 계산된 V_{ij} 와 V_{ij-1} 의 거리 값을 통해서 프레임을 구성하는 객체들의 변환에 사용되는 MVP 행렬들의 변화량을 얻을 수 있다.

그림 3.5 의 알고리즘 1-14단계를 통해 연결 리스트 L_i 를 이용하여, API 호출의 분석을 통해 i 번째의 프레임을 구성하는 *Object ID* 와 상응하는 MVP를 저장한다. 단계 10-11 에서와 같이 결과적으로 현재 프레임과 이전 프레임 간에 구성하는 객체들 사이에서 D_{ij}를 통해 구해진 MVP 변화량들 중 가장 큰 변화량인 D_{max}를 구한다. 그리고 스크린 공간의 최대값을 기준으로 정규화 하여 D_{nor} (단계 12)을 얻는다. 이후 계산된 D_{nor} 은 사람의 인지 능력 기준으로 정의된 프레임 비율 과 변화량 상관관계를 참조하여 최종적으로 높은 해상도와 낮은 해상도의 렌더링 비율인 P_i 를 얻는다.

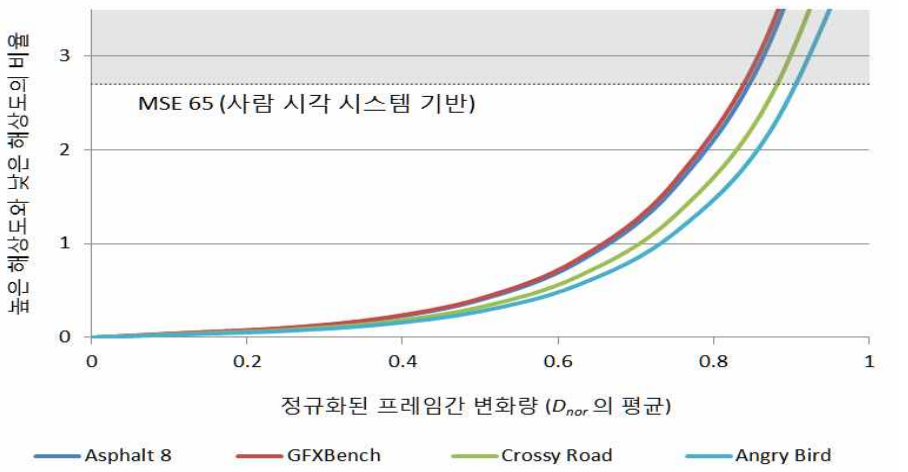
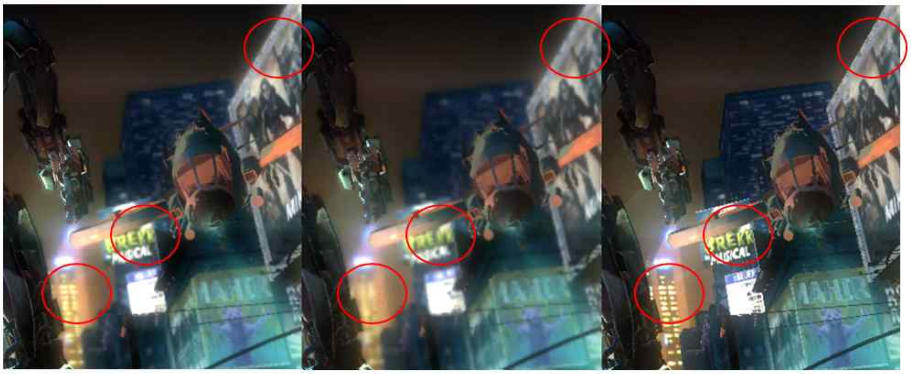


그림 3.6 높은 해상도와 낮은 해상도의 비율과 변화량과의 그래프

그림 3.6 은 프레임 간 변화량이 사람의 인지 능력 관점에서 어느 정도의 높은 해상도와 낮은 해상도의 렌더링 비율이 유효한지를 보여준다. 화질 열화는 사람의 인지 능력관점에서 정량화된 기준으로 MSE(Mean square error)를 통해 표현하였다. 6장의 실험 환경 및 결과 와 성능 분석에서 자세히 다루겠지만, 사용자 입장에서 경험할 수 있는 다양한 시나리오를 크게 2가지로 분류하였다. 렌더링 필요 연산이 하드웨어 성능 보다 높아 디스플레이 동기화 신호 기준을 만족하지 못하는 경우와, 반대로 하드웨어 성능이 필요 연산보다 높아 디스플레이 동기화 신호 기준보다 더 빠르게 렌더링하는 경우이다. 그리고 각각의 경우에 사용자와 상호 작용이 많은 경우와 프레임 간 변화량이 차이를 가지고 대표 응용프로그램 4개를 선정했다.

사람 시각 시스템(Human visual system: HVS)[10] 기준 낮은 해상도와 높은 해상도의 렌더링 비율이 임계 치를 넘으면 사람의 인지 능력 관점 그래픽 품질을 만족하지 못하는 것을 확인할 수 있다. 예를 들어, D_{nor} 의 값이 GFXBenchmark의 경우 0.814, Asphalt 8 의 경우 0.811 이상일 경우 높은 해상도와 낮은 해상도의 비율이 1:3 이 가능하나, 이 경우 HVS 기준을 만족하지 못하기 때문에 유효하지 않다. 참고로, 만일 프레임 간의 변화량 관점에서 프레임과 프레임 사이에 객체가 새롭게 추가되거나 없어지면, 이 경우는 프레임 변화가 가장 많이 일어난 경우로 간주하여 D_{max} 는 1.0으로 계산되며, 낮은 해상도는 HD수준의 해상도인 1280x720을 적용하였다.

3.3.3 그래픽 품질 개선 스케일링



(a) (b) (c)

그림 3.7 화질 개선 스케일링을 통한 동적 렌더링의 보상
(a)원본 이미지 (b) 낮은 해상도 렌더링의 일반적 스케일링 결점 (c) 제안하는 화질 개선 스케일링

이미지에서 선명도(Sharpness)는 이미지의 품질을 판단하는데 매우 중요한 척도로 사용된다. 이미지에서의 글자, 텍스처, 경계선 그리고 정적인 객체들에 대해서 뭉개짐 현상이 두드러질 경우 결점으로 간주한다. 그림 3.7과 같이, 해상도 변경을 통한 동적 렌더링으로 낮은 해상도의 프레임을 스케일링 했을 경우, (b)와 같이 글자나 정적인 객체들의 경계선에서 뭉개짐 현상이 과도하게 발생할 수 있다. 이러한 낮은 해상도 프레임 렌더링 이후 스케일링 할 때 발생 가능한 문제점을 개선하기 위해 제안하는 화질 개선 스케일링 기법을 적용한다. 낮은 해상도로 GPU가 렌더링 시에 텍스처, 글자 혹은 경

계선을 보다 선명하게 렌더링하기 위해서, 낮은 해상도로 GPU가 렌더링 할 때, 텍스처 밉맵(mipmap) 레벨 조정이나 안티 엘리어싱 레벨 조절과 같이 최적의 렌더링 옵션들을 설정한다. 낮은 해상도의 렌더링을 위해 각각의 자세한 프로세스는 다음과 같다.

첫째로, 보다 선명한 이미지를 얻기 위해서 멀티 샘플 안티 엘리어싱 비율(Multi sample anti aliasing rate)을 낮춘다. 이 접근 방식은 렌더링 된 결과에서 엘리어싱 현상이 보다 두드러지는 현상이 존재하겠지만, 스케일링으로 인하여 이러한 현상은 보상된다. 둘째로, 텍스처 밉맵 레벨(Texture mipmap level)을 조절한다. 밉맵이 가장 확대된 최상위 레벨 level 0 과 그 다음 레벨인 level 1로 텍스처의 밉맵 레벨을 조정하게 되면, 가장 선명한 이미지를 얻을 수 있다 [41]. 결과적으로 가장 선명한 두 레벨로부터 얻어진 텍스처의 합성을 통해 이미지의 텍스처를 얻게 된다. 보다 정규화 된 식으로 표현하기 위해 가중 계수가 적용된 f 는 증폭인자 $f(L)$ 의 함수라 하고, 여기서 L 은 LOD로 Level of detail 을 나타낸다. 이 방정식은 가장 상위의 텍스처 레벨에서 텍셀의 색상 T 을 얻어 오며, T_{sharp} 는 하기와 같이 표현될 수 있다.

$$T_{sharp} = (1 + f(L))T_0 - f(L)T_1$$

여기서 T_{sharp} 는 새로운 텍셀의 색상이며, T_0 는 레벨 0 에서의 텍셀 색상을 의미하며, T_1 은 레벨 1에서의 얻어오는 텍셀의 색상을 의미한다. 레벨 0과 레벨 1만을 고려하는 이유는 당연히 가장 상위 밉맵

인 레벨 0 과 그 다음의 레벨인 레벨 1이 가장 확대되어 있는 텍스처로서, 가장 선명하고 어떠한 필터도 적용되어 있지 않기 때문에, 가장 날카로운 이미지를 얻을 수 있다. 함수 *f(L)* 은 *LOD* 값을 이용하여, 가중 계수 0에서 1사의 값을 만든다. 그러므로 위의 식을 통해 낮은 해상도 프레임을 렌더링 할 때 가장 선명한 이미지를 얻을 수 있다. 당연히 이러한 접근 방식은 지나치게 선명한 이미지 효과로 원래 이미지가 왜곡 될 수 있으나, 스케일링을 통해 상쇄되어 결과적으로 가장 원본 가까운 이미지를 얻게 된다. 추가 적으로, 언제나 가장 확대된 큰 텍스처를 쓰기 때문에 추가적인 메모리가 필요하다. 하지만, 증가되는 메모리 요구량은 매우 적으며, 또한 요즘 모바일 그래픽스 환경에서는 기본적으로 ASTC[49], PVRTC[43], ETC2[44] 등 다양한 텍스처 압축 알고리즘을 적용하기 때문에 문제가 되지 않는다.

제 4 장

프레임 속도 기반 최적화

4.1 프레임 보간

표 4.1 최근 연구된 프레임 보간 기법과 적용 기술

연구	정방향 재 투영 방법	역방향 재 투영 방법	폐색 영역 처리
프레임 보간 기법	적응적 그리드 기반 이미지 와핑 기법[26]	인페인팅 기법 적용 재 투영 기법[32]	인페인팅 기법 적용 재 투영 기법[32]
	이미지-공간 검색 기반 양방향 재 투영 기법[29]	이미지-공간 검색 기반 양방향 재 투영 기법[29]	
	메시 기반 양방향 재 투영 기법[37]	메시 기반 양방향 재 투영 기법[37]	

프레임 보간은 널리 비디오 인코딩에 적용되었고 시간의 중복성을 이용한 기술로서 두 개의 연속된 프레임 사이에 중간 프레임(Immediate frame)을 생성하여 보다 높은 품질의 영상을 얻는 기술이다. 여기서 표 4.1는 최근 연구들을 중심으로 적용된 가능한 프레임 보간(Frame interpolation) 기술들을 기준으로 분류하여 보여준

다. 프레임 보간 기법은 기본적으로 재 투영 기법을 기반으로 한다. 재 투영 기법은 이전 프레임과 다음 프레임의 일관성을 이용하여 이전에 계산된 데이터를 저장, 추적 그리고 예측을 통해 중간 프레임을 생성한다. 현대의 그래픽스 하드웨어에서 가장 일반적인 방법은 히스토리 버퍼, 페이로드 버퍼 혹은 케쉬에 뷰 포트 사이즈만큼의 오프 스크린 버퍼를 관리하여, 이전 프레임의 보여 지는 부분을 저장한다. 그리고 중간 프레임이 생성될 때, 버퍼 안의 데이터는 다음 프레임의 프레임 움직임을 참조하여 자신의 새로운 위치로 재 투영을 통해 중간 프레임을 생성한다. 재 투영을 위한 하드웨어가 지원함에도 불구하고, 재 투영 기법은 여전이 계산 비용 관점에서 상당히 어려운 작업이다.

이번 절에서는 먼저 일반적으로 다수의 응용 프로그램에서 사용하고 필요에 따라 서로 동시에 사용하는 두 개의 재 투영 전략에 대해서 설명한다. 두 프레임 사이에 회전, 확대 또는 변형 등 임의의 장면 전환이 있을 수 있다. 그래서 완전한 1:1 픽셀 맵핑은 존재하지 않는다. 이 경우 맵핑은 크게 두 가지 방법으로 정방향 재 투영 방법과 역방향 재 투영 방법으로 가능하다. 정방향 재 투영 방법을 사용하게 되면, 캐시의 모든 픽셀에서 시작하여 새 프레임에서의 위치를 추적하여 쓰게 된다. 반대로 역방향 재 투영 기법을 사용하게 되면, 새로운 프레임의 각 픽셀에서부터 시작한다. 즉 새로운 프레임의 각 픽셀에 대해서 캐시에서 상응하는 픽셀을 찾아 해당 값을 가져온다. 추가적으로 이전 프레임의 보여 지는 데이터에서 존재하지 않는 가려진 지역(Occlude region)을 재 투영 할 때, 예측을 통

해 가려진 부분을 채우는 방법에 대해서도 설명한다. 마지막으로 재 투영 기법의 계산비용 관점에서 한계점을 분석 한다.

4.2 정방향 재 투영 기법

재 투영 기법의 가장 기본적인 시나리오는 렌더링 된 이전 프레임의 데이터를 이용하여 새로운 프레임을 생성하는 것이다. 정방향 재 투영 기법은 생성하고자 하는 프레임의 픽셀의 새로운 위치를 캐시에 저장된 프레임으로부터 맵핑하여 투영한다. 이를 위해 각각의 픽셀에 대해서 정방향 벡터(Forward motion vector) 혹은 변이 벡터(Disparity vector)가 필요하다. 백터를 고려한 정방향 투영은 1:1 맵핑이 아니기 때문에, 새로운 프레임의 픽셀은 다수의 입력을 받거나 혹은 아예 입력을 받지 못하기 때문에 투영된 이미지에 구멍이나 틈이 생길 수 있다.

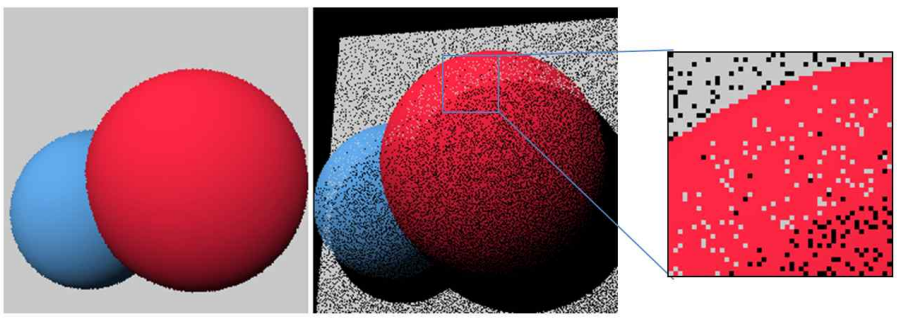


그림 4.1 정방향 재 투영 기법과 발생 가능한 문제점

그림 4.1 에 f_t 프레임에서 검은 부분이 화면의 변화로 인하여 가려지는 부분이 생긴 것을 확인 할 수 있다. 또한 재 투영 이후 앞의 붉은 색 구에도 검은 구멍이 생기는 것을 확인이 가능하며, 하얀색으로 표시된 픽셀의 경우는 고립된 잘못 된 픽셀들을 표현한다. 이러한 잘못된 픽셀들을 수정하기 위해 복잡한 추가 필터링 연산을 필요로 한다. 추가적으로 정방향 투영을 위한 연산에는 산란 연산 (Scattering operation) 이 필요로 하며, 연산 비용이 크며 느린 단점을 가지고 있다.

이러한 문제점들을 개선하는 방법 중 하나는 이미지 와핑 방법 [26]으로, 전통적인 GPU상에서 효율적으로 작동한다. 이러한 와핑은 구간적 선형으로 가정된 모션 벡터 필드를 저해상도의 그리드 상에서 근사 예측을 통해 수행된다. 초기의 정규 그리드는 이전 프레임 상의 큰 모션 벡터의 불연속점들에 의해 이동된다. 다음으로, 이 그리드 상의 기하 데이터는, 이와 연관된 텍스처가 자동으로 와핑될 수 있도록, 모션 벡터 필드에 의해 영향을 받은 새로운 위치로 그려진다. 차폐와 불연속은 그리드를 접었다 폄으로써 자연스럽게 다루어진다. 이 때, 깊이 검사는 차폐와 접힘을 정확하게 해결하기 위해 반드시 수행되어야만 한다.

이러한 이미지 와핑 기법에 의해 사용된 정규 그리드는 세밀한 기하 데이터의 이미지를 와핑하는 데 어려움이 있다. 그래서 그들은 적응적 그리드를 사용하는 개선된 알고리즘[27]을 연구되었다. 이 방

법은 정규 그리드(32x32)로부터 시작하여, 기하처리 쉐이더가 이 그리드 안의 모든 사각형을 병렬로 탐색하도록 한 후, 불연속점을 포함하는 모든 사각형을 4개로 나눈다. 이 과정은 더 이상 나뉠 사각형이 없을 때까지 반복된다. 이 때, 적응적 그리드 상의 기하데이터는 정규 그리드와 동일한 형태로 그려지게 된다. 이 새로운 방법은 적응적 그리드 덕분에 더 효율적으로 연산 자원을 사용할 수 있으며, 이렇게 함으로써 이미지의 품질을 상당히 개선시키는 결과를 가져온다.

유사한 접근 방법으로, GPU상의 병렬 데이터 산란 연산 기능을 이용 하는 정방향 재 투영 방식[28]을 제안했다. 이 방법은 캐시 상의 각각의 픽셀에 대해서, 각 픽셀의 현재 위치를 정방향 모션 벡터(불일치 벡터)를 이용해 상쇄함으로써 대상 프레임 상에서의 새로운 위치를 결정한다. 다음으로 가시성 판단을 위해, 대상 픽셀 상에서 현재 픽셀의 깊이 값이 검사된다. 이러한 연산은 병렬 쓰기 충돌을 피하여 수행된다. 단 원본과 대상 픽셀들 간의 일대일 매핑은 되지 않기 때문에, 앞서 설명한 것과 같이 재 투영 이후 구멍이 생길 수 있다. 이를 해결하기 위해, 재 투영된 픽셀의 지원 크기를 증가시키는 방법[28]을 제안하였는데, 이는 각각의 재 투영된 위치의 이웃 픽셀들에 쓰기 위함이다. 이 방법은 광원 필드를 만드는 응용프로그램을 위한 가까운 시점에서의 와핑에서는 잘 동작하지만, 비 정규화된 모션 벡터와 관련된 다른 어플리케이션에서는 비효율적일 수 있다.

최근에, 산란 연산을 활용하는 기법 보다는, 픽셀 개더링(Gathering)과 같은 전통적인 GPU 픽셀 쉐이딩 기능을 이용하는, 정방향 재 투영을 위한 이미지 기반 방법[29]을 제안하였다. 이 방법의 핵심은 렌더링 된 프레임들 상에서 대응되는 픽셀로 이어지는 모션 벡터를 찾기 위해 대상 프레임 상에서의 각 픽셀 상에서 독립적으로 수행되는 반복적인 이미지-공간 검색이다. 이 방법은 이미지 상에서 구간 적으로 부드럽다는 가정에 기반을 둔 재 투영 연산자 $\pi_{t-1}(p)$를 효과적으로 도치시킨다. 불연속성은 몇 가지 추가적인 검색 초기화 휴리스틱 알고리즘에 의해 다루어진다. 이러한 전체 과정은 픽셀 쉐이더에 적합하고, 적은 허용 시간 안에 확실한 결과를 만들어낸다.

4.3 역방향 재 투영 기법

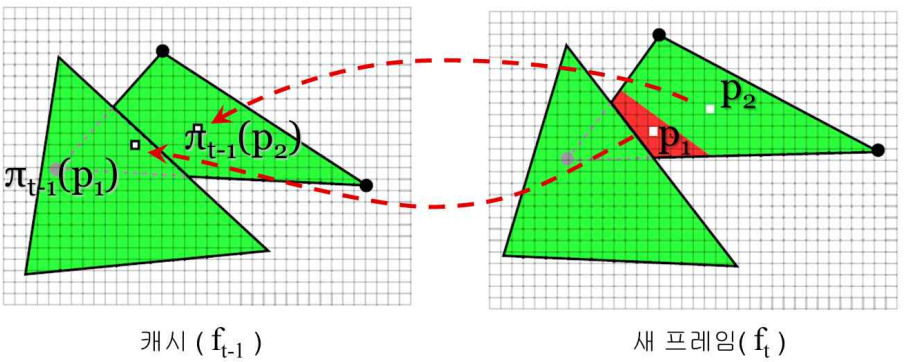


그림 4.2 역방향 재 투영 기법 및 캐시 누락

정방향 재 투영 기법과는 반대로, 새로운 프레임의 각각의 픽셀은 이미 캐시나 지정된 버퍼에 저장된 프레임으로부터 각 픽셀의 위치를 추적한다. 만일 추적한 픽셀 값이 저장된 캐시나 버퍼에 유효한 값이라면, 재사용이 가능하나, 그렇지 않다면 비싼 비용으로 다시 연산을 해야만 한다. 이러한 기술을 위한 캐시 구조를 역방향 재 투영 캐시(Reverse reprojection cache: RRC)[30, 31]라고 한다.

보다 논리적이고 정규 화된 설명을 위해 f_t 를 시간 t 에 생성된 프레임에서 보여지는 픽셀들을 저장하고 있는 프레임 버퍼라고 하자. f_t 와 함께 저장된 프레임을 구성하는 장면들의 깊이 값을 저장하고 있는 추가 적인 버퍼를 d_t 라고 하자. 여기서 $f_t(p)$ 와 $d_t(p)$ 는 픽셀 p 에 대한 각각 버퍼의 값들을 나타낸다. 그럼 시간 t 의 각 픽셀 $p=(x, y)$ 은 시간 $t-1$ 에 생성된 프레임의 화면 공간 상 위치를 가지고 추적하며 $(x', y', z') = \pi_{t-1}(p)$로 표현한다. 여기서 재 투영 연산자인 $\pi_{t-1}(p)$ 는 프레임 $t-1$ 에서 이전 위치로 픽셀 p 를 맵핑한다. 이러한 재 투영 연산에서 프레임 $t-1$ 에서 생성된 픽셀의 z' 값을 d_t 버퍼를 참조하여 얻어온다. 이렇게 얻어진 깊이 값은 현재의 픽셀이 이전 프레임에서 보여 지는 영역인지 아닌지를 확인하기 위해 사용된다. 만일, 재 투영된 깊이 값 z' 이 $d_{t-1}(x', y')$과 같은 경우, 현재 픽셀 p 와 재 투영된 픽셀 $f_{t-1}(x',y')$은 실질적으로 화면 상 동일한 지점으로 투영되는 것을 의미한다. 이 경우 이전 값은 재사용이 될 수 있다. 그렇지 않다면, 재 투영 연산의 결과는 $\pi_{t-1}(p) = \varnothing$ 이며, 픽셀 p 에 대응되는 픽셀이 없다는 것을 의미한다. 그림 4.2 에서 역방향 재 투영 동작 과정을 표현 하였다. 시간 $t-1$ 의 쉐

이딩 결과와 픽셀의 깊이 값이 화면 공간의 프레임 버퍼에 저장된다(왼쪽). 시간 t 의 각 픽셀 p (오른쪽)에 대해서 t-1 프레임의 상응하는 픽셀의 위치에 $\pi_{t-1}(p)$ 연산에 의해 재 투영된다. 투영된 픽셀의 깊이 값은 저장된 픽셀의 깊이 값과 비교되며, 동일하면 캐시 적중 p_1, 그렇지 않으면 캐시 누락 p_2을 보여준다.

4.4 폐색 영역 처리 및 한계

위에서 살펴본 재 투영 과정은 본질 적으로 비선형 와핑이며, 가려지는 부분에 대해서 잘못된 픽셀이나, 구멍이 생길 수 있다. 역방향 재 투영은 캐시 누락의 경우가 바로 추가 적인 처리가 필요한 경우로, 이러한 영역을 다시 쉐이딩하여 처리가 가능하다. 하지만, 이는 모바일 환경에서의 제한된 자원이나 실시간 처리를 고려한다면, 선택적인 옵션이 아니다. 정방향 재 투영 방식의 경우는 일반적으로 이와 같은 옵션마저도 없다. 그러므로 어떠한 형태든 재 투영 기법으로 발생할 수 있는 그래픽 결점을 해결해야 할 필요가 있다.

위의 문제점을 해결하기 위해 다양한 기법들이 제안되었고, 그 중 하나로 인페인팅 기법[32]이 제안되었다. 이웃된 이미지들을 통해서 가려진 부분을 옵셋 값을 조절하여, 4방향으로 반복적으로 수행하여 복원한다. 이 기법은 픽셀 쉐이더 관점에서 매우 효율적으로 구현이 가능하며, 구멍이 채워질 때까지 반복적인 수행을 통해서 해결 된다. 하지만 이러한 기법을 통해 결점을 개선하더라도, 채워야

할 영역이 큰 경우, 즉 빠른 큰 객체의 움직임으로 프레임이 구성되었을 때, 보간 된 픽셀은 흐리게 나타날 수밖에 없으며, 이러한 부분은 그래픽 결점으로 인지 된다. 더욱이 폐색 영역 처리 기법은 연산 비용이 매우 크다. 저전력의 모바일 환경 상에서는 적용이 근본적으로 불가능하기 때문에, 제한된 자원과 실시간 처리 관점에서 새로운 접근 방식의 연구가 필요하다.

4.5 인간 시각 시스템 기반 홀드-타입 뭉개짐

프레임 보간 기법을 통해 중간 프레임을 생성함에 있어서, 대부분의 연구는 어떻게 보다 정확한 픽셀을 계산하여 생성함에 있었다. 본 논문에서 핵심 관점인 사람의 인지적 능력을 고려한 프레임 보간 기법에 활용하기 위해 홀드-타입 뭉개짐(Hold-type blur)을 이용한 최근 연구들을 살펴본다.

과거, 화면을 지우고 갱신하고 하는 형태가 아닌 오늘 날의 LCD는 오랜 시간 동안 화면에 뿌려주는 홀드-타입 디스플레이를 대부분 사용한다. 그 결과 이러한 홀드-타입 디스플레이에서 사람의 인지 능력 관점을 활용한 다양한 연구 들이 진행되고 있다. 화면상에 빠르게 움직이는 객체의 경우 사람의 눈이 이 객체를 따라가면서 보기 때문에 뭉개져 보이는 현상이 존재한다. 뭉개짐 인식 현상에 대한 연구가 많은 부분에서 과거 잘못 분석되었었고, 많은 연구들을 통해 뭉개짐을 인식되는 원인에 대해서 대부분은 홀드-타입 뭉개짐

현상에 의해 기인되는 것이 증명 되었다[33, 35]. 특히 낮은 프레임 속도에서 이 효과는 극대화 될 수 있으며, 결과적으로 그래픽 품질의 현저한 저하를 확인할 있다. 또한 사용자 응답속도도 느려지게 된다[26].

그래픽 렌더링 관점에서 보면, 프레임 보간의 문제는 상당히 간단하게 설명된다. 버퍼로 주어진 프레임을 렌더링 할 때, 정확한 픽셀의 속도(Velocity)와 기하 정보를 추출할 수 있기 때문이다. 이에 따라, 이미지 기반의 추측을 통해 단점을 개선할 수 있다. 추가적으로 만일 여러 장의 중간 프레임을 생성한다고 가정했을 때, 각각 모두의 중간 프레임을 키 프레임으로 정확하게 동일한 품질로 생성할 필요는 없다. 이 개념을 활용하여, 중간 프레임은 낮은 품질로 생성하여 활용하는 기법[26]을 제안했다. 즉, 사람의 인지적 관점에서 충분히 높은 프레임 속도로 이미지가 보여 지고 있으면, 인간의 시각은 어느 수준 이상의 프레임 속도의 프레임들은 뭉개져 보이게 된다는 점[36]을 활용하였다. 그러므로 중간에 생성된 프레임의 결점들이 빠른 프레임 속도로 유지가 될 경우, 사람의 인지 관점에서는 가려질 수 있다.

여기서, 사람의 인지 능력관점에서 프레임 간 빠르게 움직이는 객체가 많을 경우 일정 속도 이상의 프레임 속도에서는 뭉개져 보이는 점을 착안하였고, 이 특성을 활용하여, 프레임 간 변화량 기반 가변 해상도에 낮은 해상도로 렌더링 할 경우 발생할 수 있는 뭉개짐 효과의 결점이 보상가능 하다는 가정을 하게 되었다.

다른 접근 방법으로, 시간적 업 샘플링을 기반으로 한 이미지 기반 근사 와핑 기법[32]을 통한 프레임 보간 방법을 제안하였다. 이 연구는 높은 프레임 속도를 가정하지 않고, 30fps로 렌더링 되는 낮은 프레임 속도에서 60fps로 업 샘플링 하는 경우를 고려하였다. 해당 연구의 이미지 와핑 방식은 단 방향의 재 투영만을 고려하여, 프레임 간의 정적인 객체들에 대해서는 좋은 결과를 가져오나, 빠르게 움직이는 동적인 객체들에 대해서는 재 투영 기법으로 인한 결점들이 나타난다. 비록 제안한 기법에 결점들을 해결하기 위해 하나 이상의 프레임을 참조하여 프레임을 보간 하도록 하였지만, 이로 인한 추가적인 비용과 프레임 출력의 지연으로 실시간 환경과 모바일의 저전력 환경에서는 적합하지 않다.

이후 단 방향 재 투영 방법으로 인한 결점들을 보안하기 위해 양 방향 방식으로 재 투영하는 기법[29] 이 제안되었다. 보다 진화된 재 투영 기법으로 렌더링 된 두 개의 프레임을 양 방향 재 투영을 통해 중간 프레임을 보간 하여 생성한다. 이 방법을 통해 프레임의 생성 지연도 효과적으로 개선이 가능하고, 단 방향 재 투영 방식 보다 정점-바운드(Vertex-bound)와 픽셀-바운드(Pixel-bound) 응용 프로그램에서 전반적으로 높은 성능 향상을 확인 할 수 있었으며, 음영 처리나 모션 뭉개짐(Motion blur)들이 많이 포함되어 있는 경우에서도 결점이 많이 개선된 것을 확인할 수 있었다. 이후 이 방법을 개선하여, 메시 기반 기법[37]으로 개선하였다.

하지만, 이러한 프레임 보간 방식은 이미 GPU의 자원을 다 쓰고 있는 상황이라면, CPU나 다른 연산처리 장치를 통해 처리 할 수밖에 없다. 추가적으로 비록 성능이나 연산량 측면에서 개선되었다고 하나, 근본적으로 모션 벡터 연산과 모션 벡터 기반 재 투영 기법이기 때문에 상당한 추가 연산을 필요로 한다. 이러한 추가 연산은 곧 전력 소모의 결과를 초래하며, 저전력 환경의 모바일에서는 적용할 수가 없다. 이를 해결하기 위해 모션 벡터 기반이 아닌 새로운 접근 방식인 타일 기반 GPU를 활용한 중간 프레임 전달 방식을 통한 프레임 보간 기법[34]을 다음 절에서 제안한다.

4.6 타일 기반 GPU의 전력 소모 최적화 및 품질 개선

최근 모바일 기기에서 높은 품질의 멀티미디어의 요구사항이 높아지고 있으나 모바일 GPU의 하드웨어적 성능, 전력, 온도 등의 문제로 서비스 제공에 한계가 있다. 이러한 한계로, 높은 연산량이 요구되는 그래픽 응용프로그램의 경우, 높은 사용자의 응답성과 높은 수준의 그래픽 품질을 보장할 수 없다. 추가적으로, 소모 전력과 온도와 같은 모바일 기기들의 고질 적인 문제로 일부 프레임 처리를 무시하는 등 강제로 품질 감소 방식을 통해 처리한다. 이러한 방식은 끊어지거나(Flickering) 또는 늦게(Lagging) 출력되는 현상 등의 그래픽 감소를 유발한다.

앞서 언급했듯이 이를 극복하기 위한 다양한 프레임 보간 기법 기반의 프레임 속도 증가 기법(Frame rate up conversion: FRUC) 들이 연구되었다. 빠른 사용자의 응답성과 높은 수준의 그래픽 품질을 보장하기 위해 최근 까지 연구된 프레임 속도 증가 기법은 재 투영과 모션 벡터들을 기반으로 한 프레임 보간 기법이 가장 널리 사용되었다. 이러한 프레임 보간 기법은 예측을 통한 재 투영에 기초를 두고 있기 때문에 앞 장에서 설명한 다양한 결점들이 나타난다. 비록 기존의 기법들의 결점들을 해결하기 위해 다양한 연구들이 이루어져 왔지만, 해결하기 위한 추가 비용과 잠재되어 있는 프레임 출력의 지연으로 실시간 환경과 모바일의 저전력 환경에서는 적합하지 않다.

기존의 프레임 보간 하는 방식을 이미 GPU의 자원을 다 쓰고 있는 한계상황을 고려해 보자. 프레임 증가를 통한 서비스 품질의 향상을 위해서는 GPU가 아닌 CPU나 다른 연산처리 장치를 통해 처리 할 수밖에 없다. 앞서 설명한 바와 같이 단방향과 양방향 기반의 프레임 보간 기법들이 비록 성능이나 연산량 측면에서 개선되었다고 하나, 자원의 한계를 벗어나는 높은 추가 비용을 통한 해결 방법은 모바일 환경에서는 유효하지 못하다. 이러한 추가 비용은 곧 높은 전력 소모와 온도 증가 문제를 초래하며, 저전력 환경의 모바일에서는 적용할 수가 없다. 이를 해결하기 위해 성능, 전력 그리고 온도의 제한이 있는 모바일 기기에 적합하도록 추가 적인 높은 비용이 없이 동일한 연산량으로 타일 기반의 GPU를 고려한 새로운 접근 방식의 프레임 보간 기법을 제안한다.

4.6.1 타일 기반 렌더링

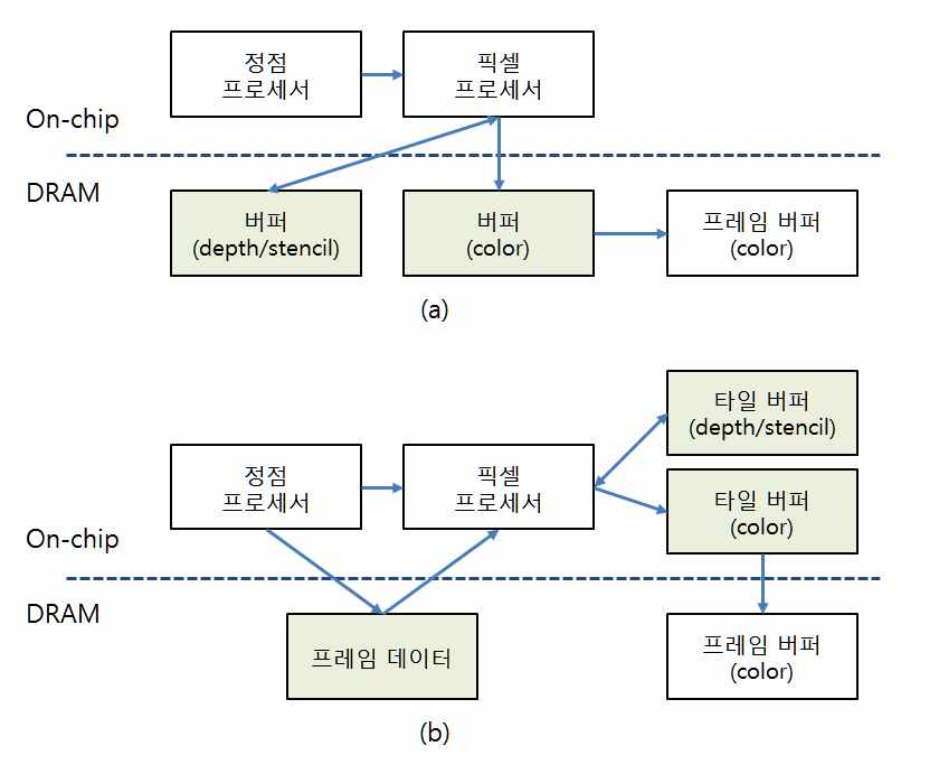


그림 4.3 즉시 모드 렌더링과 타일 기반 렌더링 방식의 구조적 차이

PC의 GPU는 기본적으로 즉시 모드 렌더링(Immediate-mode rendering: IMR) 구조로 구현되어 있다[1]. 즉시 모드 렌더링에서는 프레임의 장면의 객체들을 묘사하는 기하 데이터(Geometry)는 정점 프로세스의 의해서 변환되고, 다음 객체의 정점들을 처리하기 전에

즉시 픽셀 프로세스를 위해 그래픽 파이프라인에 전달되고, 프레임 버터에 써진다. 만일 다음에 오는 객체가 이전에 그려졌던 객체보다 화면상 앞에 있을 경우, 픽셀처리를 위한 연산을 다시 해야 하며, 곧 프레임 버퍼에도 다시 써져야 한다. 이러한 문제는 일반적으로 덮어쓰기 문제[76]로 하나의 객체들의 깊이 관계와 처리하는 순서의 불일치로 픽셀을 처리하기 위해 여러 번 추가 연산을 해야 하고, 주 메모리에 여러 번 써져야 한다. 결과적으로 메모리 대역폭의 낭비를 초래한다. 하나의 프레임에 하나의 픽셀의 써지는 횟수를 종종 깊이 복잡도[47] 로 언급되기도 한다.

그림 4.3에서는 즉시 모드 렌더링과 타일 기반의 렌더링 방식의 구조적 차이를 보여준다. 타일 기반 렌더링(Tile-based rendering: TBR) 구조[8] 는 덮어쓰기 문제와 메모리의 대역폭을 해결하기 위해 설계되었다. 타일 기반 렌더링에서는 화면을 타일 이라는 픽셀의 직사각형 블록으로 분할하고, 각각의 타일은 렌더링 되어 렌더링 된 타일들이 모여 하나의 완성된 프레임이 된다. 타일은 주 메모리가 아닌 칩 안의 메모리(On-chip memory) 에 처리 및 저장될 수 있도록 충분히 작게 설계되어 주 메모리의 접근을 최대한 줄여 대역폭을 절감한다.

즉시 모드 렌더링과는 다르게 변환된 객체의 정점 데이터들은 최종적으로 프레임 버퍼에 직접 써지는 것이 아니라, 주 메모리의 장면 버퍼[46]에 저장한다. 그리고 객체의 정점들은 기본적으로 삼각형 단위로 처리 되며 타일 단위로 처리된다. 타일에 포함되어 있는 객

체들의 삼각형들은 깊이 값에 따라 처리되어, 모든 기하 데이터들이 처리 되고 타일단위로 렌더링이 모두 완료 되면, 최종 렌더링 된 타일들은 주 메모리의 한 번의 접근을 통해 써진다. 즉, 칩 내부 메모리에서 타일 단위로 기하 데이터의 처리와 렌더링을 하고 각 타일들이 렌더링이 완료되었을 때, 칩 외부 메모리에 접근하여 프레임 버퍼에 상응하는 위치에 써지어, 최종적으로 모든 타일들이 다 써지면, 해당 프레임의 렌더링이 완료된다. 그러므로 타일 기반 렌더링은 최종 렌더링 결과를 쓰기 위한 외부 메모리의 상당한 메모리 대역폭을 절감할 수 있다. 하지만, 변환된 삼각형들의 정보는 외부 메모리에 저장되어야 하고 렌더링을 위해 메모리부터 가져와야 한다. 결과적으로 픽셀 프로세스를 위한 메모리 대역폭은 감소할 수 있지만, 추가 적인 정점 처리를 위한 메모리 대역폭은 요구되어 진다.

여기서 중요한 점은 메모리의 실질적인 대역폭은 픽셀 프로세스에 의한 대역폭이 주요 원인이며, 타일 기반 렌더링이 저전력 시스템에서 메모리의 총 대역폭의 큰 감소[48]를 확인할 수 있다. 비록 아직도 즉시모드 렌더링이 PC 기반의 GPU부문에서 지배적이지만, 정점 처리를 위한 추가 메모리가 상대적으로 픽셀 처리를 위해 발생되는 메모리 대역폭 보다 매우 적기 때문에 타일 기반 렌더링이 모바일 부문에서 보다 적합하며 주로 활용되는 기술이다[70, 77, 53]. 이러한 타일 기반의 렌더링 방식을 활용하여 사용자의 응답속도를 높이고 GPU의 전력 감소 및 프레임 속도를 최대 2배까지 올리는 새로운 방법을 제안 한다.

4.6.2 중간 프레임 전달 기법 기반 프레임 속도 증가

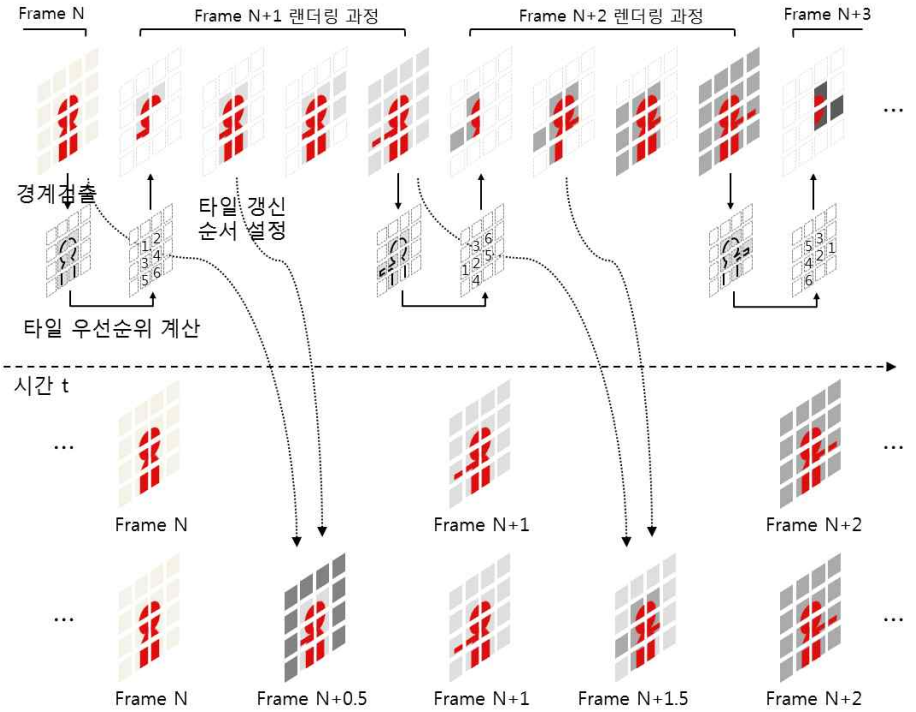


그림 4.4 중간 프레임 전달 방식을 통한 중간 프레임 생성

기존의 프레임을 기반으로 중간 프레임을 생성하는 방식으로 프레임 보간 방법은 프레임 내의 객체 움직임(Motion vector)를 추정하고 이를 이용하여 새로운 중간 프레임을 생성하는 방식이다. 하지만, 연산량이 많고 이에 따른 전력 소모와 온도 증가 문제로 전력 소모에 제약이 있는 모바일 기기 등에는 적합하지 않다. 따라서 성

능, 전력, 온도에 제약이 있는 모바일 기기 등에 적합하도록 중간 프레임을 생성하는 새로운 접근 방식인 타일 기반 중간 프레임 전달 방식(Half frame forwarding: HFF)을 통해 중간 프레임을 생성하는 기법[34] 을 제안한다.

그림 4.4은 타일 단위로 화면을 나누어서 출력하는 타일 기반GPU를 활용한 프레임 속도 높이는 방법(Frame rate up-conversion: FRUC)의 개념 이해를 위한 흐름을 보여주고 있다. 현재 대부분의 모바일 GPU는 타일 기반 렌더링 방식을 사용 중이다. 타일 기반 렌더링은 프레임 전체를 한 번에 렌더링하지 않고 타일 단위로 분할하여 처리하는 방식으로 메모리 대역폭, 데이터 캐싱등 다양한 측면에 있어 효율이 높은 특징이 있다. 타일 기반 렌더링의 경우 타일 단위로 최종 픽셀까지 GPU 내부에서 렌더링 후 외부 메모리 (예를 들어 DRAM) 에 할당된 프레임 버퍼에 저장한다.

위와 같은 모바일 GPU의 타일 기반 렌더링을 활용하여 이전의 렌더링된 타일에서 경계검출을 통해서 동적인 영역을 추출하여, 해당 영역에 포함되어 있는 타일에 새로운 우선순위를 할당하여, 앞으로 렌더링 할 프레임의 타일 갱신 순서를 조정한다. 이러한 조정을 통해서 현재 프레임이 렌더링이 완료되기 절반의 시점에서 이전 프레임과 합성을 통해 중간 프레임을 생성한다. 제안하는 프레임 속도 증가 기법의 가장 핵심적인 아이디어는 최종 프레임에 대한 렌더링이 완료되지 않았더라도 이미 처리가 완료된 타일들에 포함된 픽셀은 최종 데이터이며, 이 타일 내 픽셀 데이터를 활용한다는 점이다.

이미 렌더링 완료된 *N* 프레임과 현재 렌더링 중인 *N+1* 프레임의 중간 결과 픽셀 데이터를 이용하면 중간 *N+0.5* 프레임을 지연(Lagging) 없이 생성 가능하다.

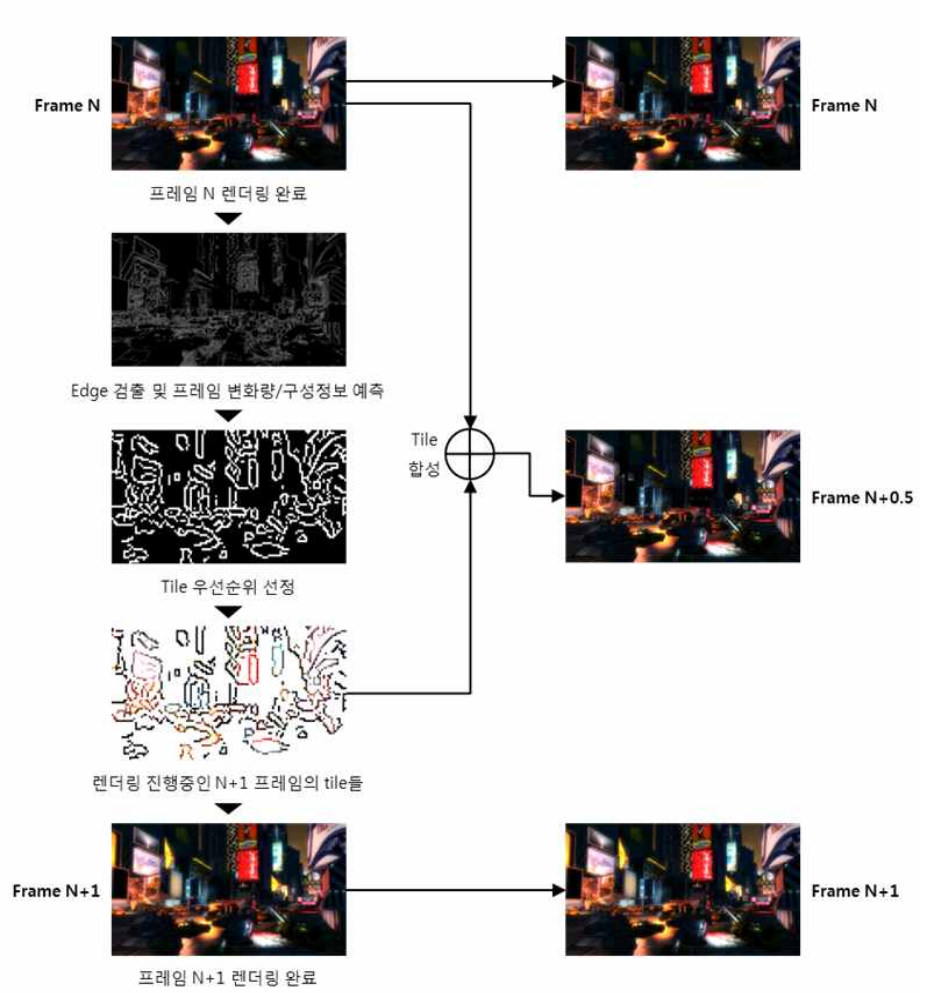


그림 4.5 경계 검출을 통한 동적 타일들의 합성 흐름

제안하는 프레임 속도 높이는 방법은 중간 프레임을 생성하기 위해 앞서 설명한 타일 기반의 렌더링의 특성 기반으로 이미 완료된 프레임과 생성 중인 중간 결과를 합성하여 중간 프레임에서 발생하기 때문에 타일간의 단절 현상이 발생할 수 있다. 이러한 문제점을 해결하기 위해 동적 객체들의 타일을 우선적으로 처리해야 하는 필요성이 있다. 동적 개체로 인한 화질의 열화는 주로 경계를 중심으로 발생되고 이러한 경계가 포함되어 있는 타일을 찾기 위해 주로 경계 검출(Edge detection) 알고리즘을 적용하였다.

그림 4.5는 렌더링 된 N 번째 프레임을 기반으로 동적 개체가 포함되어 있는 타일을 얻어, 합성하는 주요 알고리즘의 과정을 설명한다. 핵심 알고리즘은 경계 추출 기법을 통해 얻어진 렌더링 된 N 번째 프레임의 경계 정보를 기반으로 프레임 내의 오브젝트가 움직일 가능성이 있는 모션의 양을 예측한다. 첫 단계로 렌더링이 완료된 N 번째 프레임에서 경계 정보를 추출한다. 둘째 단계로 경계 정보를 분석하여 프레임을 구성하는 내용을 단순 도형(Solid figure), 복잡도형(Complex figure) 그리고 글자(Text) 등으로 구분한다. 셋째 단계로 프레임에 대한 모션 예측 정보와 구성정보를 기반으로 다음 프레임의 타일들의 렌더링 우선순위를 지정하여, 지정된 타일 렌더링 우선순위를 GPU에 업데이트 한다. GPU는 업데이트 정보를 기반으로 타일 렌더링을 실행하게 된다. GPU가 타일 렌더링 할 때, 처리된 타일의 총 개수가 적응적 임계값을 넘었을 경우, 또는 프레임을 출력하여야 할 시점에 도달하였을 경우 이전 프레임과 처리된 타일들을 합성(이전 프레임에 현재 렌더링 된 타일 내 픽셀 데이터

를 합성)하여 중간 프레임을 생성한다. 생성된 중간 프레임을 디스플레이에 출력한다.

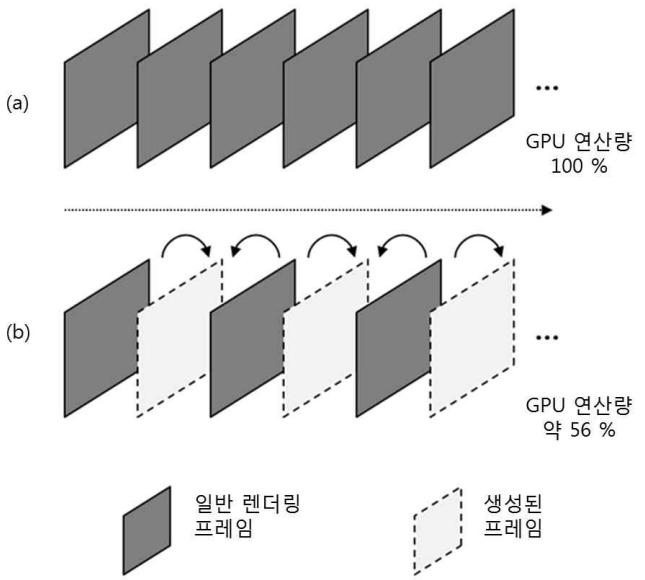


그림 4.6 제안한 프레임 속도 증가 기법을 통한 GPU 연산량 감소
(a) 60fps 렌더링 (b) 30fps 렌더링 + 제안된 FRUC 기법

제안하는 프레임 속도 높이는 방법은 크게 두 가지의 시나리오를 통해 GPU의 전력감소 혹은 사용자 응답성 측면과 그래픽 품질 측면에서 높은 서비스의 제공이 가능하다. 먼저, 그림 3.13에서는 그래픽 연산량이 작은 응용 프로그램의 경우 모바일 GPU 하드웨어는 1초에 60장 이상의 렌더링이 가능하다고 가정하고 디스플레이 장비는 60Hz로 동작한다고 가정하자. 이 경우, 1초에 30장만 렌더링하여, 제안하는 프레임 속도 증가 기법을 통해서 60장의 프레임의 결

과를 얻을 수 있다. 즉, 60장을 그리기 위한 GPU의 연산량을 30장의 연산인 절반 정도로 줄일 수 있다. 당연히, 경계 검출을 위한 연산 비용이 추가 적으로 필요하지만, 해당 비용은 렌더링하는 비용에 비해서 매우 적은 비용이다. 실험을 통한 자세한 분석 내용은 6장에서 설명한다. 다음으로 그래픽 품질을 향상시키기 위한 방법으로 활용될 수 있다.

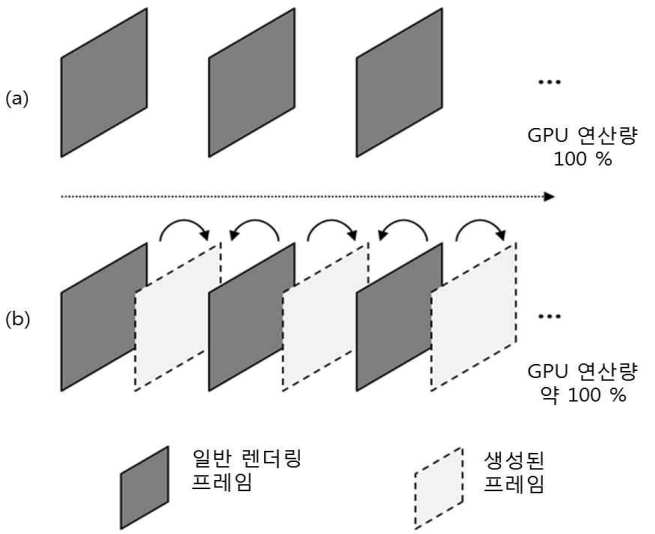


그림 4.7 제안한 프레임 속도 증가 기법을 통한 서비스 품질 향상
(a) 30fps 렌더링 (b) 30fps 렌더링 + 제안된 FRUC 기법

그림 3.14에서와 같이 그래픽 연산량이 매우 높은 응용프로그램의 경우 GPU의 성능은 1초에 60장 이하로 렌더링하며, 이 경우 앞서 설명한 끊김 현상과 사용자 응답시간의 지연과 같은 문제가 발생할 수 있다. 마찬가지로 디스플레이 장비는 동일하게 60Hz로 동작한다

고 가정하자. 제안하는 프레임 속도 증가 기법은 기존의 이동 벡터를 구하고 구해진 벡터(Motion vector)기반 복잡한 이동 보상 보간 방식(Motion Compensate Interpolation)의 프레임 증가 방법의 문제점들을 극복하여 매우 적은 비용인 경계 검출을 통해 서비스 품질을 향상 시킨다. 추가적으로 향후 VR[42] 등 120Hz기반의 성능을 현재의 낮은 성능의 하드웨어로 유사한 품질의 서비스를 제공할 수 있다.

4.6.3 인간 시각 시스템 기반 프레임 분석

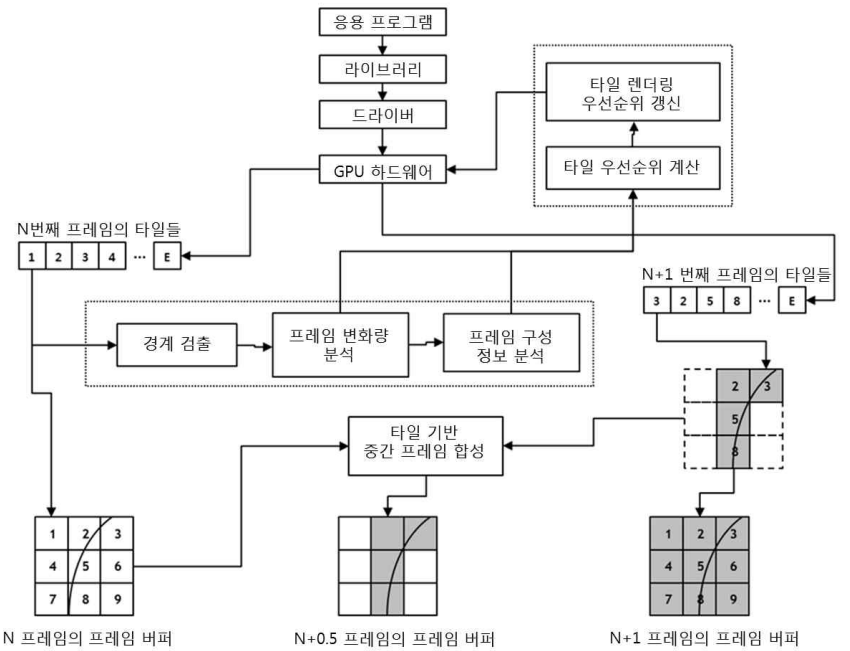


그림 4.8 경계 정보 기반 중간 프레임 생성 알고리즘 흐름

그림 4.8에서는 제안하는 알고리즘의 주요단계의 흐름을 묘사하고 있다. GPU 하드웨어로부터 렌더링 된 결과 N 번째 프레임으로부터 경계 검출을 통해 동적인 영역을 추출한다. 추출된 동적인 영역 기반으로 전후 프레임간의 변화량을 분석하고, 프레임의 구성 정보를 분석한다. 이렇게 분석되어진 정보를 바탕으로 다음 렌더링 될 프레임의 타일의 우선순위를 계산하여, 새롭게 할당될 타일의 갱신 순위를 GPU 하드웨어에 업데이트를 한다. 이렇게 갱신된 타일 우선순위 정보를 기반으로 현재 프레임을 타일 단위 렌더링 한다. 즉, 프레임간의 동적인 부분을 먼저 우선적으로 렌더링하는 것이다. 이렇게 우선순위 타일 기반 프레임이 렌더링이 모두 완료되기 전 시점에 이전 프레임 N 번째 프레임과 합성하여, 현재 프레임의 렌더링이 완료되기 전인 $N+0.5$ 인 중간 프레임 생성이 완료된다. 결과적으로 진행 중인 프레임의 렌더링이 완료되기 전 중간 프레임을 지연 없이 생성이 가능하다. 참고로, 앞서 시스템 개요에서 설명 했듯이, GPU의 디바이스 드라이버를 통해 타일 기반 GPU 하드웨어의 타일 갱신 순서를 제어가 가능하다는 가정을 기반으로 한다.

그림 4.9는 프레임 간 변화량과 구성 정보를 예측하는 단계를 상세히 보여주고 있으며, 크게 3가지 세부 단계로 구성되어 있다. 프레임의 경계 정보를 추출하는 단계와 추출된 경계를 기반으로 전후 프레임 간의 변화량을 예측하는 단계 그리고 프레임의 구성 정보를 예측하는 단계이다. 경계 추출 단계는 렌더링 된 프레임의 픽셀 데이터를 입력 받아 픽셀 기반의 경계 검출(Edge detection) 을 수행하는 단계이다. 프레임은 타일 단위로 렌더링 되며 우선 렌더링 된

타일 결과 데이터를 먼저 처리하여 지연 없이 경계 검출 할 수 있다. 시스템 환경에 따라 추가적인 버퍼를 활용하여 복수의 타일 단위로 경계를 추출할 수 있으며, 여기서는 추가 적인 버퍼 사용에 대해서는 다루지 않는다.

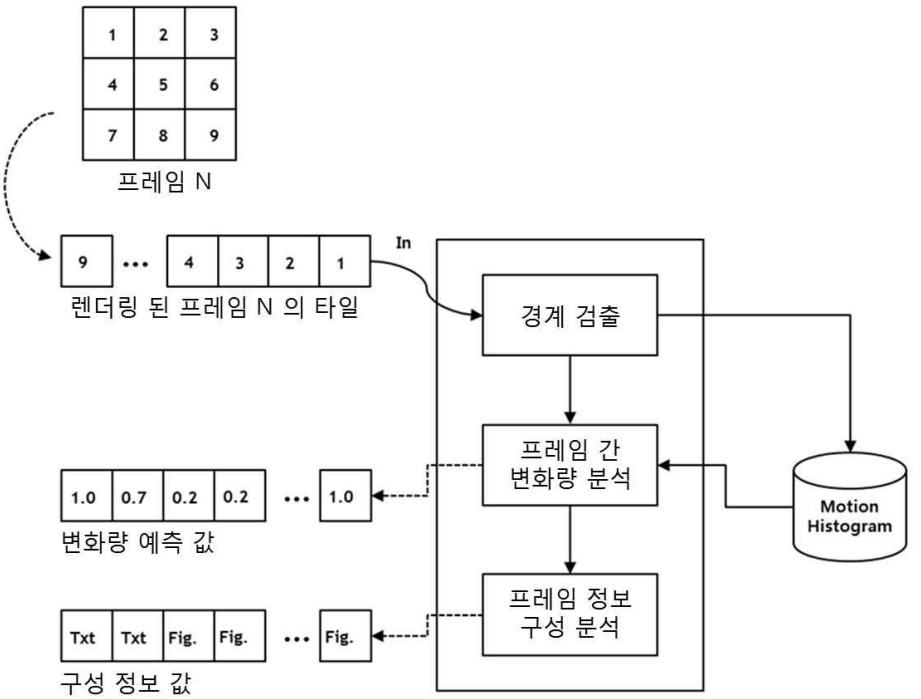


그림 4.9 경계 정보 기반 프레임 변화량 구성 정보 예측

전후 프레임 간의 변화량 예측(Motion estimator) 단계는 타일 인덱스(Index) 별로 기록된 경계의 히스토그램(Histogram)을 이용하여 변화량을 예측한다. 매 프레임마다 N 번째 프레임의 타일과 이전

N-1 번째 프레임의 동일 인덱스를 가지는 타일에서 검출된 경계 변화량과 해당 인덱스의 타일의 변화량 히스토그램간의 상대적 차이를 비교하여 프레임 영역 별 변화의 정도를 예측한다. 프레임 구성 정보(Object information) 예측 단계는 타일 내에 수직, 수평 경계 정보량을 분석하여 렌더링 된 타일의 결과물을 단순 구성(Solid figure), 복잡한 구성(Complex figure) 혹은 글자 기반(Text) 등의 유형으로 분류한다. 분류된 정보는 저장되며 이는 타일 우선순위 선정에 활용된다. 또한, 저장된 정보는 중간 프레임의 생성 비율을 유연하게 조절하는데 활용 가능하며, 예측된 프레임 간 변화량이 적을수록, 프레임 구성이 복잡할수록, 글자의 구성이 적을수록 높은 비율을 통해 중간 프레임을 생성한다. 하지만 여기서는 프레임 생성 비율은 고려하지 않고, 한 장 만을 생성하는 경우로 기존 프레임 속도의 최대 두 배까지 증가하는 경우만을 고려한다.

4.6.4 렌더링 우선순위 계산 및 합성

타일의 렌더링 우선순위를 계산 단계는 프레임 간의 변화량 정보와 프레임 구성 정보를 입력 받아 우선순위를 결정하는 단계이다. 수평성분 경계의 경우 프레임 안의 객체가 수평 이동할 경우 수직 이동 대비 타일 변화량이 많을 가능성기 높기 때문에 해당 경계가 포함된 타일의 좌우 타일에 대한 우선순위 갱신이 필요하다.

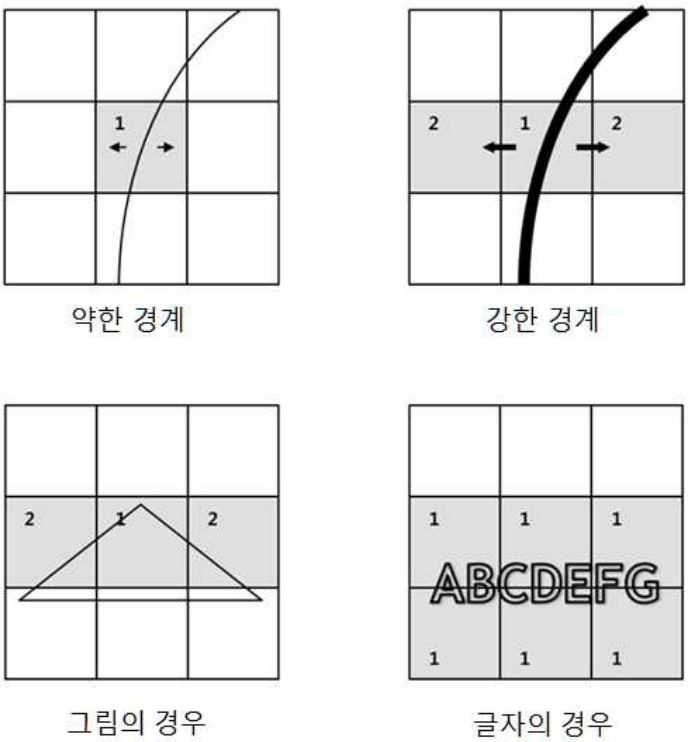


그림 4.10 경계 검출 및 분석에 따른 타일의 우선순위 설정

그림 4.10에서는 우선순위를 지정할 좌우 범위는 경계의 강도에 따라 가변적으로 지정되는 것을 보여준다. 우선순위는 원 위치에서 멀어질수록 이동하였을 확률이 낮아지기 때문에 비례하여 우선순위도 낮아지게 된다. 수직성분 경계의 경우는 수평 성분에 대한 지정방식과 동일하되 범위를 상하 타일에 대해 지정하게 된다. 타일 내에 수평성분 경계와 수직성분 경계가 복잡하게 뒤섞인 경우 글자(Text) 유형으로 판별한다. 글자의 경우 일부 타일만 업데이트 할 경우 의미를 잃을 수 있기 때문에 포함된 모든 범위를 지정하고, 각 타일에

대해 동일하게 우선순위를 지정한다. 이후, 타일 렌더링 순서를 다음 프레임 처리를 위해 GPU에 갱신하는 단계로서, 타일 렌더링 우선순위 계산 단계에서 얻어진 우선순위를 현재 렌더링 진행 중인 프레임에 갱신 하는 장치이다. 이미 처리 진행 중인 타일을 제외하고 나머지 타일들에 대한 렌더링 우선순위를 다시 지정하며 이때 GPU는 렌더링을 멈추거나 지연시키지 않는다.

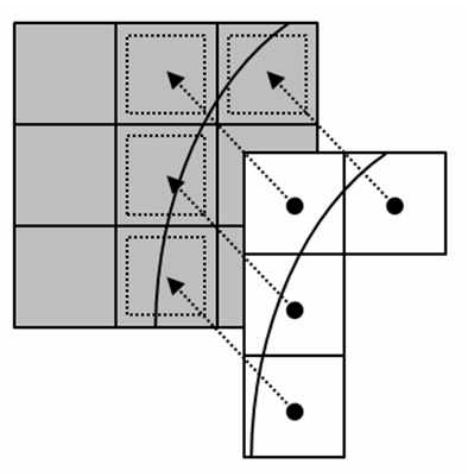


그림 4.11 타일 합성

그림 4.11은 중간 프레임 생성을 위한 타일 합성으로 이전에 처리된 *N-1* 번째 프레임의 데이터와 렌더링 진행 중인 *N* 번째 프레임의 타일들의 결과물을 합성하는 예를 보여준다. 타일 합성은 크게 두 단계로 이루어진다. 타일의 우선순위의 갱신에 의하여 합성이 되어야 하는 타일의 여부를 확인한다. 다음으로, 합성이 되어야 하는 경우라면, 동일한 인덱스 타일간의 프레임 변화량 비교 결과를 확인하

여, 합성을 통한 갱신 혹은 생략하도록 한다. 결과적으로 모든 프레임은 복수의 타일 단위로 세분화되어 처리되며 처리 완료된 각각의 타일들은 프레임 버퍼에 저장된다. 처리된 N 번째 프레임의 타일의 총 개수가 시스템에서 정의된 임계값에 도달하였을 때, N 번째 프레임의 렌더링 지연 없이 N-1 번째 프레임의 데이터와 합성을 처리한다. 합성 시 동일 인덱스의 타일간의 프레임 변화량이 일정 비율 이상 차이 날 경우 해당 타일에 대한 합성을 하지 않는다. 합성으로 인한 오류를 줄이기 위한 주요한 경우로 급격한 밝기 변화, 장면에 객체 추가 혹은 삭제, 카메라 시점의 완전 전환이 대표적인 예이다.

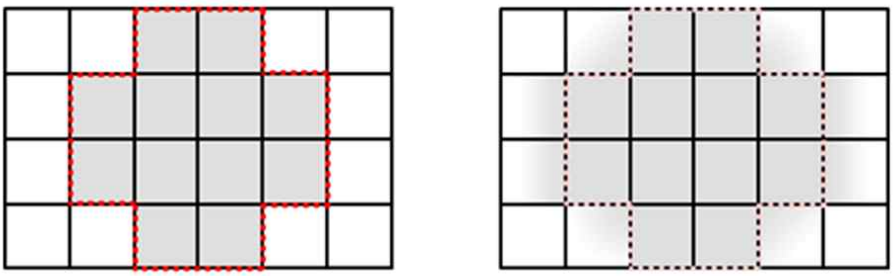


그림 4.12 타일 경계의 단절 문제 및 개선

마지막으로, 그림 4.12와 같이 타일 합성 시 타일의 경계면에서 단절로 인한 추가 적인 화질 열화(픽셀의 색상, 이동 등의 급격한 차이 등으로 인한 절단면)가 발생 할 수 있다. 이를 개선하기 위해 다양한 방식의 픽셀 간 보간 기능을 지원하며 이는 타일로부터 분석된 변화량 정보에 유연하게 선택/해제 가능하며, 시스템 호출을

통해 시스템 외부에서 사용자가 필요 혹은 목적에 따라 최적화가 가능하다.

제 5 장

데이터 재사용을 통한 최적화

5.1 데이터 재사용

그래픽스 관점에서 데이터 재사용의 가장 일반적인 방법은 이전 프레임의 쉐이딩 결과를 이용하여 현재 프레임의 연산량을 줄이는 방식이다. 다시 말해, 재사용을 통해 렌더링 시 요구되는 픽셀의 갱신을 줄이는 방법을 말한다. 이 방법은 일반적으로 각 픽셀에 대한 공간과 시간 개념에서의 다수의 샘플의 조합을 통해 적분과 같은 높은 연산의 태스크의 연산을 분할하여 연산량을 상쇄시킨다. 데이터의 재사용의 기본 개념은 공간과 시간의 일관성을 이용한 샘플링을 통해 프레임 간 시간적 중복성과 공간적 중복성을 최적화함으로서, 요구되는 GPU의 연산량을 줄이는 것이다. 시간적 중복성 외에도, 예를 들어 저주파 디퓨즈 쉐이딩과 같은 경우는 공간적 중복성이 존재한다. 대부분 기존 연구들은 공간 혹은 시간적 중복성 하나에 대해서만 연구가 진행되었지만, 시간적 중복성 뿐 아니라, 공간적 중복성을 함께 활용하는 연구[12]가 제안 되었다.

실제로 하나의 픽셀을 처리하기 위한 비용은, 이전 데이터로부터

주어진 프레임을 계산하기 위해 요구되는 샘플링 된 픽셀들을 통한 연산 비용이 훨씬 적으며, 유사한 그래픽 품질의 이미지를 얻을 수 있다. 이러한 이전의 여러 프레임들로부터 시간 공간의 일관성을 기초로 샘플링 하여 현재 프레임의 고품질 렌더링 효과를 적은 연산 비용으로 구현이 가능하다.

가장 최근 발표된 OpenGL ES 3.x[62]를 통해 다양한 기능을 지원하는 모바일 기기들이 많이 상용화되고 이를 활용한 높은 수준의 렌더링 효과가 적용된 그래픽 콘텐츠가 늘고 있다. 그럼에도 불구하고, 대표 기능중 하나인 멀티 렌더 타깃(Multi-render target: MRT)[64]에 대한 데이터 재사용은 연구된 적이 없다. 특히, 멀티 렌더 타깃은 한 장의 최종 렌더링 결과를 위해 동시에 여러 타깃으로 분할하여 렌더링하기 때문에, 이미 분할된 정보를 기반으로 시간적 일관성을 이용하여 데이터 재사용이 매우 용이하다. 다음 절에서 시간적 일관성을 활용한 데이터 재사용을 통한 멀티 렌더 타깃의 데이터 재사용 기법[45]에 대해 자세히 설명한다.

5.2 멀티 렌더 타깃의 재사용을 통한 최적화

최근 모바일 기기들을 활용하는 높은 품질의 멀티미디어의 보급이 늘어남에 따라, GPU 하드웨어의 성능 제약이 심화되었다. 특히 높은 수준의 다양한 렌더링 효과를 적용하기 위해서는 높은 연산 비용이 요구된다. 앞서 살펴 본 바와 같이 최근까지 이미 렌더링 된

여러 프레임으로부터 시간 및 공간의 일관성을 활용하여 샘플링을 통한 데이터 재사용 기법이 연구되었다. 이러한 연구를 통해 고품질 렌더링 효과를 적은 연산 비용으로 구현이 가능하며, 공간 기반 안티 엘리어싱[39], 부드러운 그림자 효과[38] 그리고 글로벌 조명 효과[40] 에서 쉐이딩 연산의 감소를 통한 성능 개선 연구가 이루어졌다.

가장 최근 발표된 OpenGL ES 3.x[62] 는 위의 렌더링 효과를 포함한 추가적인 다양한 기능을 지원한다. 대표적인 예로, 폐색 쿼리 (Occlusion queries)[58], 변환 피드백 (Transform feedback)[63] 그리고 멀티 렌더 타깃 (Multiple render targets)[64] 들이 있다. 이러한 기능을 활용한 모바일 기기들이 많이 상용화되고 이를 활용한 높은 수준의 렌더링 효과가 적용된 그래픽 콘텐츠 보급도 늘고 있다. 그럼에도 불구하고, 대표 기능중 하나인 멀티 렌더 타깃(Multi-render target: MRT)[64]에 대한 데이터 재사용은 고려된 적이 없다. 특히, 멀티 렌더 타깃은 한 장의 최종 렌더링 결과를 위해 동시에 여러 타깃으로 분할하여 렌더링하기 때문에, 많은 메모리 대역폭을 요구한다. 이러한 문제점을 개선하기 위해, 이미 분할된 정보를 기반으로 시간적 일관성을 이용한 멀티 렌더 타깃에 적용 가능한 새로운 데이터 재사용을 방법을 제안[45]한다.

5.2.1 멀티 렌더 타깃

멀티 렌더 타깃 기술은 가장 최근에 제안된 기술 중 하나로서, 대표적으로 지연 쉐이딩(Deferred shading)에 활용되어 전통적인 쉐이딩(Forward shading)과 달리 객체별로 라이팅 연산(Lighting operation)을 하지 않고 전체 장면에 대한 한 번의 라이팅 연산으로 동일한 결과를 얻을 수 있으며, 연산 효율을 증가 시킬 수 있다. 멀티 렌더 타깃 기술은 GPU의 픽셀 쉐이더가 하나의 픽셀에 대한 데이터를 여러 개의 버퍼로 저장이 가능하다. 이렇게 저장된 버퍼를 높은 품질의 그래픽 효과를 위해 라이팅 쉐이더의 매개변수로 사용할 수 있다. 이러한 접근방식을 통해 라이팅 연산은 모든 지오메트리가 렌더링 된 후에 이루어지며, 멀티 패스 렌더링과 같은 부하 없이 장면 전체에 한 번의 라이팅 연산으로 처리가 가능하다. 이를 위해 저장되는 사용되는 대표적인 버퍼들은 컬러(Color), 노멀(Normal), 포지션(Position), 재료(Material)등이 있다.

그림 5.1은 멀티 렌더 타깃 기법을 통한 동적범위가 큰 화면의 글로우 효과(Glow)를 보여준다. 그림 5.1(a)-(c)는 멀티 랜더 타깃에 저장된 결과를 보여준다. 세 가지의 다른 버퍼에는 각각 순서대로, 컬러(Color), 깊이(Depth) 그리고 표준(Normal)값 쓰여 진다. 응용프로그램에 따라, 렌더링 타깃의 개수가 다르며, 보여주는 예는 3개를 고려하지만, 그림 5.2와 같이 GFXBenchmark의 경우는 디퓨즈(Diffuse), 반사(Reflection), 표준(Normal), 스팩큘러(Specular) 그리고 깊이(Depth) 총 5개의 렌더 타깃 버퍼를 두고 사용한다.

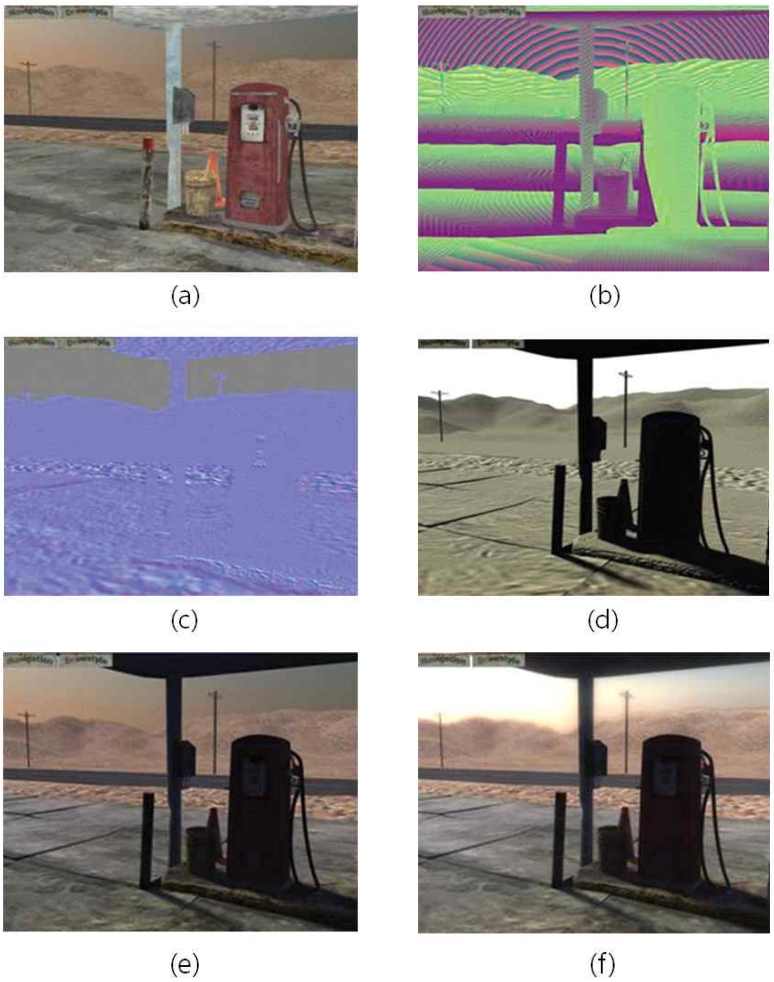


그림 5.1 멀티 렌더 타깃과 각 타깃 버퍼의 이미지와 합성

(a) 컬러 버퍼, (b) 깊이 버퍼, (c) 노멀 버퍼,
(d) 노멀 버퍼와 깊이 버퍼를 이용한 라이팅 연산 결과,
(e) 컬러 맵과 이미지의 합성, (f) 최종 이미지

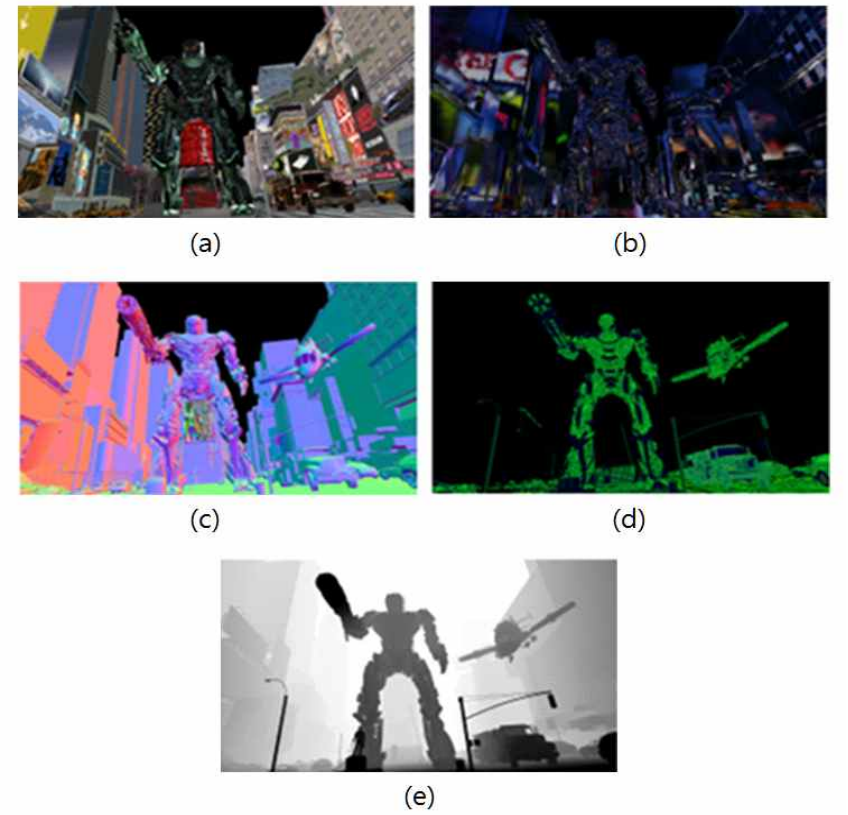


그림 5.2 GFXBenchmark에서 사용하는 멀티 렌더 타깃 종류
(a) 컬러 버퍼 (b) 반사 버퍼 (c) 노멀 버퍼 (d) 스팩큘러 버퍼
(e) 깊이 버퍼

그림 5.1 (c)-(d)는 광원의 조건에 따라, 생성된 표준 버퍼와 깊이 버퍼의 값을 통해 라이팅 계산된 결과를 보여준다. 마지막으로 그림 5.1의 (e)-(f)는 마지막 최종 단계를 보여준다. 해당 단계에서는 라이트 값이 0과 1범위를 벗어나는 고강도 빛의 값을 화면에 적용하여, 높은 명암을 가지는 영역 주위에 픽셀 블루밍(초점 번짐)효과를 포

함한 높은 수준의 라이팅 연산의 효과를 표현할 수 있다. 이러한 연산을 통해 그림 3.20 (d)와 같은 글로우(Glow) 효과가 적용된 이미지[85] 를 확인할 수 있다. 멀티 렌더 타깃을 사용함으로써, 먼저 멀티 패스의 랜더링의 부하를 줄일 수 있으며, 장면을 구성하는 모든 객체들의 복잡한 라이팅 연산을 대신하여 한 번의 라이팅 연산으로 그림 3.20 (d)와 같은 뛰어난 화질의 최종 장면을 만들 수 있다. 픽셀 쉐이더의 복잡도는 높은 품질의 다양한 그래픽 효과에 따라 증가하고 있다. 그러므로 장면에 보이지 않는 픽셀들의 불필요한 연산 비용을 줄이는 멀티 렌더 타깃을 통한 지연 쉐이딩의 구현은 필수적인 요소이다.

5.2.2 시간적 일관성을 이용한 데이터 재사용

멀티 렌더 타깃은 동시에 복수의 렌더 타깃에 렌더링하기 때문에 메모리 대역폭 관점에서 단점이 있다. 이는 스마트폰 등과 같이 메모리 대역폭에 제한이 있는 기기에 치명적이다. 이를 개선하기 위해 앞서 설명한 시간의 일관성을 이용한 데이터 재사용을 통해 연산량 및 메모리 사용량을 줄일 수 있다. 특히 과거의 재사용 기법들 중 로컬 저장 공간을 이용을 통한 방법도 제안되었으나, 이러한 기법은 추가 확장 API를 사용해야 하기 때문에, 정해진 하드웨어서만 동작하였다. 이러한 한계가 있기 때문에 일반적인 활용은 힘들다. 하지만, 제안하는 멀티 렌터 타깃에서 활용 가능한 데이터 재사용 기법은 하드웨어의 종속적이 없고, 연산의 추가 비용이 적으면서 메모리

대역폭 절감이 가능하다. 곧 이러한 결과는 GPU의 소모 전력의 감소로 이어진다.

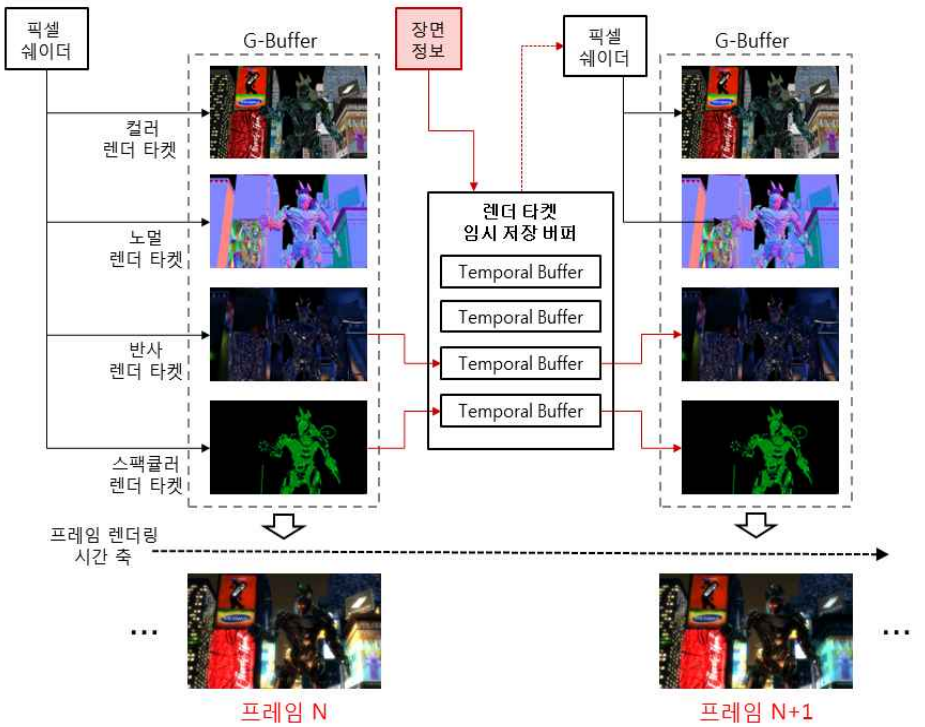


그림 5.3 멀티 렌더 타깃에서의 데이터 재사용 개념

그림 5.3은 멀티 렌더 타깃에서의 데이터 재사용 기법의 핵심 개념을 보여준다. 시간의 일관성을 이용하는 제안하는 기법의 핵심은 장면의 전환이 발생하기 전까지 전후 프레임 간 영상의 차이가 크지 않다는 점을 활용한다. 즉, 이러한 시간의 일관성 기반으로 데이터 재사용을 통해 메모리 사용을 줄이고, 연산 비용을 절감하는 기

법이다. 이를 위해 멀티 렌더 타깃을 각각의 버퍼를 저장하는 영역을 G-Buffer라고 하며, 이 영역에 저장된 렌더 타깃 중 프레임의 정보 분석을 통해 재사용 가능한 렌더 타깃을 지정한다. 지정된 렌더 타깃은 다음 프레임에서 다시 렌더링하지 않고, 다음 프레임 렌러링에 그대로 재사용한다.

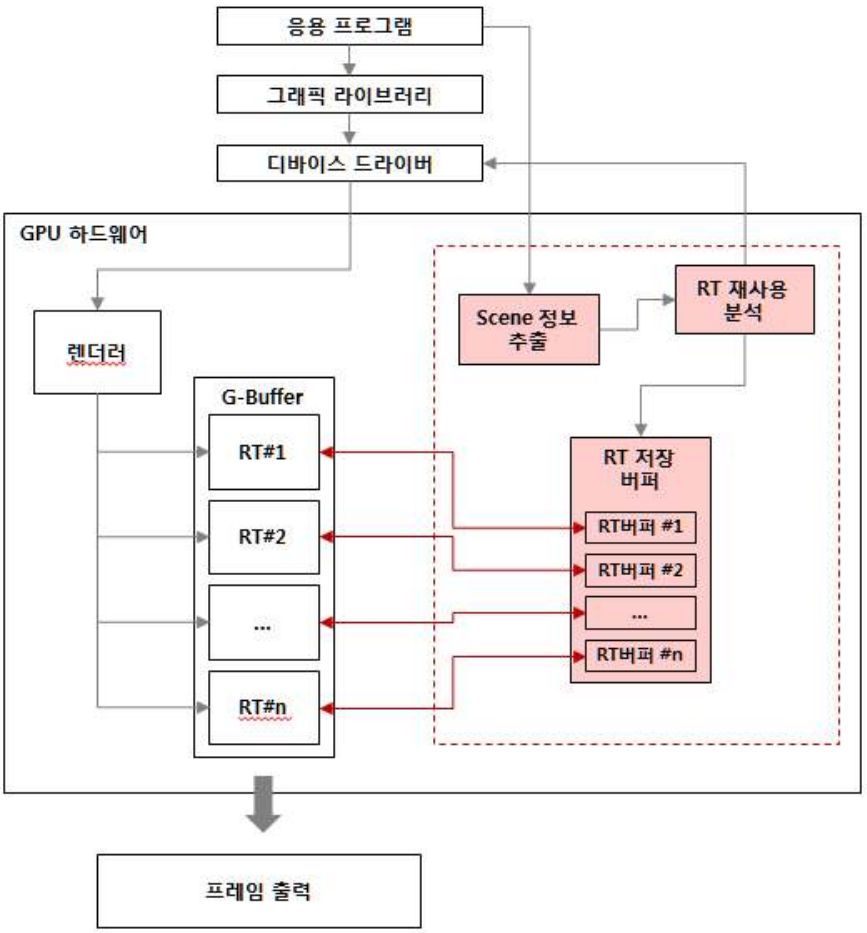


그림 5.4 데이터 재사용의 알고리즘 단계와 흐름

데이터 재사용에 있어서, 일반적으로 오브젝트 표면에서의 빛과 반사효과를 표현하기 위한 반사나 스팩큘러 렌더 타깃 등은 위치가 약간 틀어지더라도 사람이 인지하기 어렵기 때문에 우선적으로 재사용이 가능하다. 하지만, 화면의 급격한 전환이 있을 경우에는 시간의 일관성이 거의 없다. 이런 경우를 고려하기 위해 프레임 렌더링 시 프레임의 장면 정보를 분석 한다. 프레임 렌더링에 사용되는 드로우 호출 회수(Draw call count), 삼각형의 개수(Triangle count) 그리고 정점의 수(Vertex count)등을 분석하여 화면의 급격한 전환이 있을 경우는 렌더 타깃의 데이터를 재사용하지 않는다. 보다 자세한 세부적인 알고리즘 단계를 설명하기 위해 그림 5.4 는 시간의 일관성을 이용한 멀티 렌더 타깃에서의 데이터 재사용을 위한 알고리즘 흐름을 보여준다. *N* 번째 프레임의 렌더 타깃을 *N+1* 번째 프레임에 재사용 경우를 가정한다.

첫째 주요 단계로 장면 정보 추출 단계이다. *N* 번째 프레임 렌더링을 위한 데이터 구성 중 장면 정보 추출 단계에서는 화면 구성 정보를 추출한다. 이 경우, 추출된 정보와 히스토그램에서 과거 프레임의 추출된 정보 (*N-1, N-2.. N-t*)를 비교 분석한다. 둘째로 분석 결과를 바탕으로 렌더 타깃 재사용 분석 단계에서는 장면의 구성과 변화의 분석 결과를 기반으로 재사용할 렌더 타깃을 지정한다. 만일 프레임간의 변화가 적을 경우, 주어진 프레임 렌더링 시에 지정된 렌더 타깃은 렌더링하지 않고 이전 데이터를 재사용하며, 반대로 장면의 전환이 급격한 경우면, 모든 렌더 타깃은 렌더링을 통해 재사

용 하지 않고 갱신한다.

이 경우, GPU가 렌더링 된 N 번째의 프레임 렌더 타깃들이 G-Buffer 에 렌더링 되지만, 여기서 두 가지 경우로 나뉜다. 하나는 G-Buffer 에 결과가 보존되는 경우로 재사용 렌더 타깃을 따로 저장할 필요가 없고 다음 프레임 렌더링 시 그대로 활용 가능하기 때문에 추가 저장으로 인한 메모리 오버헤드가 없다. 하지만, 예외적으로 응용 프로그램이 G-Buffer 의 써진 내용을 지워(Flush) 하는 경우가 있다. 이러한 경우에 렌더 타깃 저장 장치에 지정된 재사용에 필요한 렌더 타깃은 추가적인 버퍼 할당을 통해 따로 저장해야 한다. 결과적으로 재사용 지정된 렌더링 타깃을 제외한 나머지 렌더링 타깃들에 대해 *N+1* 프레임 프레임의 GPU 렌더링 실행을 실행하고, 렌더링 된 렌더 타깃과 재사용된 렌더 타깃을 이용하여 최종 프레임을 출력한다.

5.2.3 렌더 타깃 저장

앞서 간단히 설명했듯이, 멀티 렌더 타깃의 렌더 타깃 데이터를 다음 프레임에 재사용하기 위하여 저장여부를 고려해야 한다. 기본적으로 추가적인 저장으로 인한 메모리의 오버헤드를 없애기 위해 써진 렌더 타깃의 정보는 지우지(Flush) 않는 것을 원칙으로 한다. 하지만. 특정 응용 프로그램에 의하여 강제적으로 G-Buffer가 지워지는 예외적인 상황이 존재할 수 있으며, 이러한 경우를 재사용을

위한 렌더 타깃의 저장 버퍼를 할당하여 사용한다. 추가 적인 저장 공간은 외부 메모리에 위치하며, 저장 가능한 버퍼들의 종류는 그림 5.5 에서와 같이 보여준다. 즉, G-Buffer에 저장되어 데이터를 2D 텍스처 형태로 저장하며, 컬러, 노멀, 반사, 스팩큘러 4개의 버퍼의 내용을 2D 텍스처 형태의 데이터로 저장한다.

그림 5.5 2D 텍스처 형태로 저장된 각 렌더 타깃
(a) 원본 이미지
(b) 컬러, 노멀, 반사, 스팩큘러의 2D 텍스처 형태 데이터

5.2.4 인간 시각 시스템 기반 렌더 타깃 재사용

멀티 렌더 타깃의 가장 중요한 핵심 알고리즘은 사람의 인지적 관점에서 그래픽 품질의 감소를 인지하지 못하는 범위에서 이미 써진 렌더 타깃의 데이터를 얼마나 재사용이 가능한지 결정하는 것이

다. 결과적으로 사람이 그래픽 성능의 감소를 인지하지 못하게 렌더 타깃의 데이터의 재사용을 극대화 하여 메모리 감소를 최대화 하는 것이 핵심이다.

표 5.1 재사용 정도에 따른 메모리 및 화질 변화

메모리 감소	Normal MRTs	2RT 재 사용	효율	4RT 재 사용	효율
Memory Read (MB/frame)	178	178	0.0%	178	0.00%
Memory Write (MB/frame)	132	122	7.6%	113	14.39%
Total (MB/frame)	310	300	3.2%	291	6.13%

화질 열화	Normal MRTs	2RT 재 사용	4RT 재 사용
Average MSE	0	4	120

표 5.1은 그래픽 적으로 매우 복잡한 대표 응용프로그램인 GFXBenchmark의 Manhattan 3.0[83]을 통해 재사용하는 렌더 타깃의 수에 따라 감소되는 메모리 대역폭과 원본 대비 화질 열화 정도를 보여준다. 비록 최대한 많은 개수의 렌더 타깃을 재사용하는 것이 메모리 관점에서는 가장 효율적이겠지만, 그 만큼 많은 그래픽 결점들이 발생한다.

그림 5.6 장면 전환 시점의 데이터 재사용 생략

추가적으로, 장면 정보 분석을 통해 급격한 프레임의 장면 전환 시점에 대한 고려가 필요하다. 예를 들어, 시점, 객체의 추가 및 생성, 광원의 빠른 변화는 데이터 재사용으로 인한 시각적 결점이 발생할 수 있으며, 장면 정보를 추출 단계에서 추출하는 정보는 GL API를 통하여 처리되는 가시적인 광원 효과와 3.1절에서 제안한 모델 뷰 프로젝션 행렬의 정보를 통해 가능하다. 하지만, 여기서는 화면의 가장 큰 전환 시점을 얻는 것을 목적으로 하기 때문에, 드로우 콜의 호출 회수, 삼각형의 개수, 정점들의 개수들을 통해서 그림 5.6과 같은 급격한 변화 시점 분석이 가능하다.

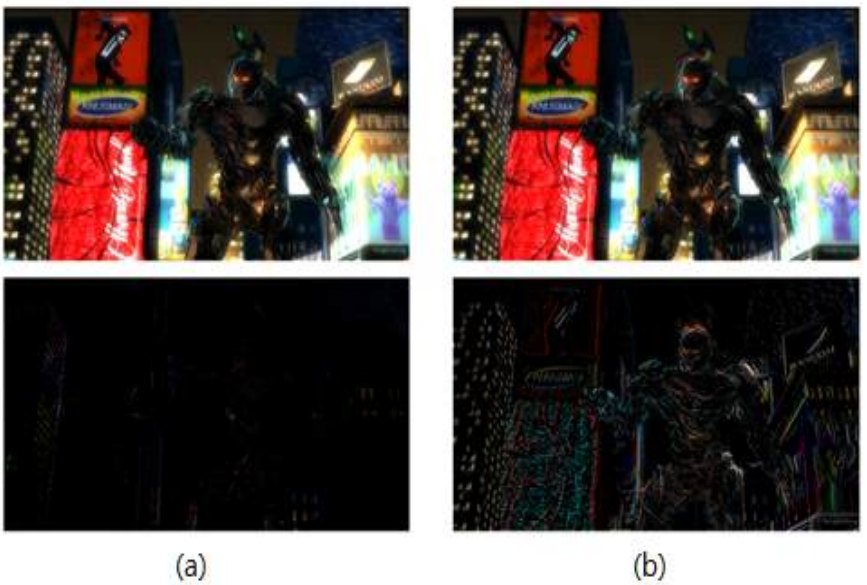


그림 5.7 렌더 타깃 재사용 정도에 따른 화질 열화
(a) 2개의 렌더 타깃의 재사용 및 원본 대비 화질 차이
(b) 4 개의 렌더 타깃의 재사용 및 원본 대비 화질 차이

그림 5.7은 2장의 렌더 타깃을 재사용했을 경우와 4장의 렌더 타깃 모두를 재사용했을 때의 원본과의 화질 차이를 보여준다. 이와 같이 언제나 최대한의 렌더 타깃을 재사용하는 것은 유효하지 못하다. 사람의 인지적 관점에서 그 임계값을 지정할 필요가 있다. 실험을 통한 자세한 분석은 사람 시각 시스템 기반의 실험 결과와 성능을 통해 다음 6장에서 자세히 설명한다.

제 6 장

성능 분석

이번 장에서는 저전력 모바일 환경에서 그래픽스 관점으로 제안한 방법들의 유효성을 검증한다. 첫째로, 성능과 GPU 전력 소모 측정을 위한 실험 환경의 설명한다. 둘째로, 그래픽 품질 평가를 위해 사람 시각 시스템(Humarn Visual System: HVS)[10] 과 평가 지표에 대해 설명한다. 이를 바탕으로, 최근까지 연구된 기법들과의 비교 분석을 통해 제안한 기법들의 성능을 평가한다. 결과적으로 제안한 기법들이 사람의 인지 능력 관점에서 그래픽 품질을 만족할 수 있다는 점을 증명함과 동시에 GPU의 연산량의 감소와 GPU의 소모 전력 감소 결과들을 통해 연구 성과를 보여준다. 마지막으로 결과 분석을 통해 저전력 모바일 환경에서 제한된 하드웨어 성능으로 높은 사용자 응답성 보장과 높은 품질의 그래픽 서비스의 제공 가능함을 보여준다.

6.1 실험 환경

6.1.1 구현 및 환경

보다 신뢰성 있는 실험 결과를 제공하기 위해, 실제 상용화된 모바일 기기 LG G3 Screen[67] 을 기반으로 실험을 진행한다. CPU는 ARM사의 빅.리틀(big.LITTLE)[74] 클러스터 기반으로 CA15와 CA7[73] 코어 각각 4개를 빅과 리틀 클러스터로 구성하여 8개의 코어로 구성 된다. 최근까지 연구된 기술들의 성능 평가의 공정한 비교를 위해 구현된 기술들은 빅 코어의 동일한 고정 성능으로 실행하며, 1.6Hz로 동작하도록 설정한다. GPU의 경우는 Imagination사의 Rogue[78]를 사용하며, 400MHz까지 동작이 가능하다. GPU 연산량을 디스플레이 장비의 동기화 신호(1/60초)안에 동작하는 GPU의 동작시간을 기반으로 성능을 측정하기 때문에 동적 전압 및 주파수 변경 기법(Dynamic voltage frequency scaling: DVFS)로 인한 변수를 제거하기 위해 DVFS는 비활성화 한다.

그림 6.1에서 3장에서 제안하는 3가지의 방법들의 구현 계층을 보여준다. 기본적으로 디바이스 드라이버 계층에서 구현이 가능하지만, 프레임 간 변화량을 통한 가변 해상도 조절 방법은 디바이스 드라이버의 종속성 없이, 미들웨어 영역의 추가 구현을 통해서도 가능하다. 응용 프로그램에서 일어나는 API 호출의 중간 미들웨어어서 처리하도록 하면, 응용 프로그램을 통해서 불리는 API를 래퍼

(wrapper)계층을 통해 각각의 프레임을 구성하는 객체들의 변화 및 이동을 위한 4x4 행렬들의 16개의 값을 읽어오는 것이 가능하다. 이렇게 읽어온 프레임 변화들의 값을 가변 해상도를 통한 저전력 기술 구현이 가능하다.

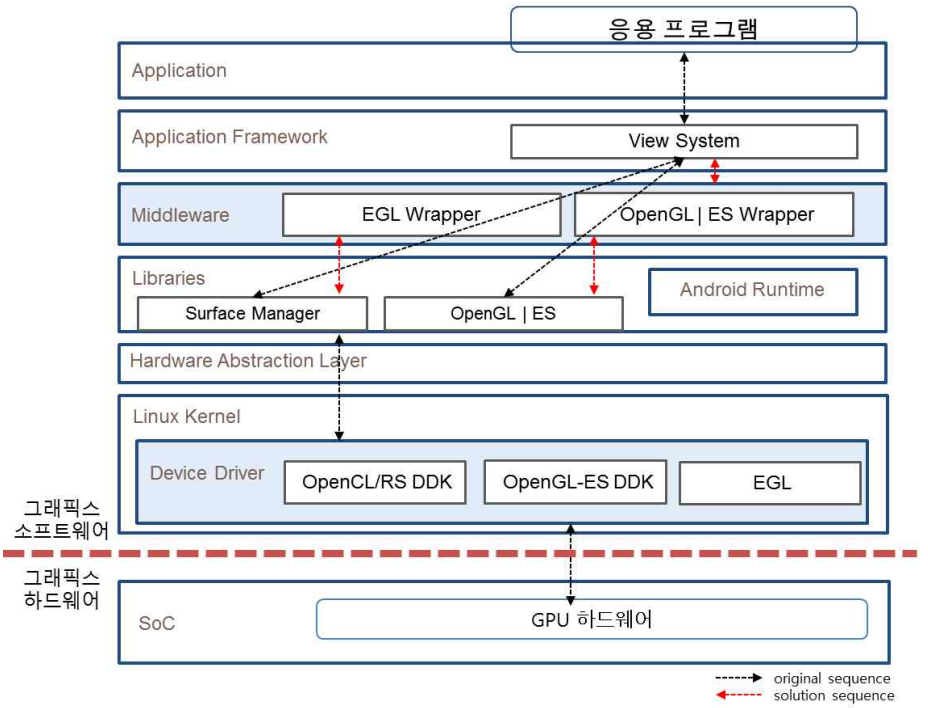


그림 6.1 제안된 기법들의 구현 계층

그림 6.2와 같이, (a)와 (b)는 LG G3 Screen 개발보드 기반의 측정 환경을 보여준다. 20개의 샘플 칩을 가지고 GPU 및 보드 전체의 전력을 측정하기 위해 DAQ NI USB-6363[79] 장비를 연결하여, GFXBenchmark 의 Manhattan 3.0 실험 벡터를 10번 반복 시행한다. 시행 마다 하드웨어 초기화는 냉각기로 시행하여 발열로 인한 오차

를 최소화 하였다. 매 주기마다 수집된 전력 소모 결과에 대한 평균을 계산하여, 전력 소모의 성능을 평가한다.

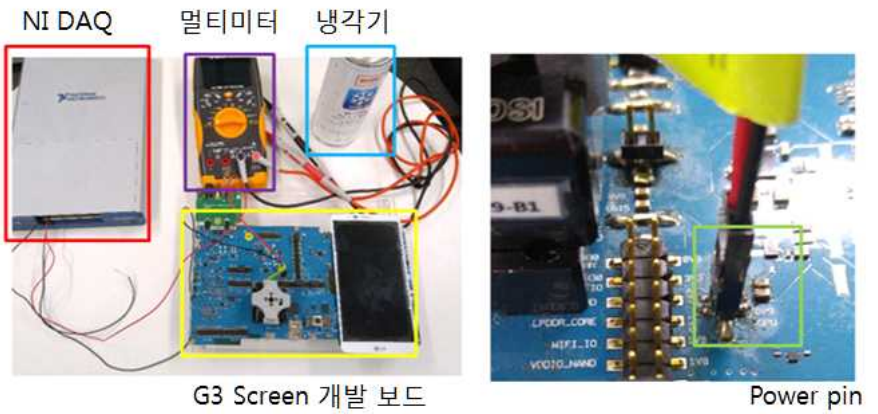


그림 6.2 전력 소모 측정 실험 환경

6.1.2 실험 벡터

실제 사용자 관점에서 다양한 응용프로그램들의 실행 환경을 재현하기 위해, 그래픽스 관점에서 크게 두 가지 시나리오로 분류한다. 다양한 시나리오를 일차 적으로, 모바일 기기의 디스플레이 환경 기준으로, GPU가 디스플레이의 동기화 신호의 속도(1/60 초)보다 더 빨리 렌더링 가능한 경우와 그렇지 못한 경우로 분류한다. 디스플레이 동기화 신호보다 빠르게 렌더링이 가능하다는 의미는 주어진 응용프로그램의 그래픽적 관점 연산량이 충분히 적어서 높은

사용자 응답 속도와 실시간 렌더링 성능을 보장가능 한 경우이다. 이러한 경우를 대표하는 응용 프로그램으로, 안드로이드 UI, Angry bird[81] 그리고 Crossy road[82]이다. Angry bird와 Crossy road 두 대표 응용 프로그램을 선택한 이유는 적은 GPU 연산을 요구하지만, 프레임 간 변화량 관점에서 하나는 많은 움직임과 다른 하나는 적은 움직임을 대표하는 경우로 사용된다.

이와 반대인 경우로, 디스플레이의 1/60초 보다 느리게 응용 프로그램이 렌더링이 되는 경우로, 요구되는 GPU의 연산량이 너무 많아 사용자의 충분한 응답 속도와 실시간 렌더링 성능을 보장하지 못하는 경우이다. 이 경우를 대표하는 응용 프로그램으로 GFX-Benchmark[83]와 Asphalt 8[80]은 가장 최근에 발표된 GPU 성능을 평가하는 대표적인 응용 프로그램들로 선정한다. 두 응용 프로그램은 현재의 모바일 GPU 하드웨어로는 60fps 의 성능을 만족하지 못하면서, GFXBenchmark의 경우는 벤치마크 응용프로그램으로서 사용자와 상호 작용이 적은 경우를 대표하며, Asphalt 8은 높은 품질의 그래픽 중심 게임으로 사용자와 상호 작용이 많은 경우를 대표한다.

6.1.3 시각 시스템 기반 화질 평가 기준

사용자 인지 능력 관점에서의 성능 평가를 위해 비디오 품질 평가에 활용되는 객관적이고 주관적인 체감품질 (Quality of

experience: QoE)[2] 평가 방법을 적용한다. 잘 알려진 주관적인 QoE 는 평균 의견 점수(Mean opinion score: MOS)로 주로 멀티미디어 응용프로그램의 품질을 평가하는데 사용한다. MOS 는 사람의 인식 능력에 기초하여 영상의 품질을 정량화하기 위한 척도로 사용한다. 사용자의 경험으로 얻은 영상 품질 척도를 통해 사용자는 영상의 등급을 통해 품질을 평가한다[2]. QoE에 대한 객관적인 측정 기준은 주어진 주관적인 품질을 정량화된 값으로 매핑하기 위한 양적 수학 모델을(Quantitative mathematical models) 통해 결정된다. 대표적인 객관적 평가 기준은 최대 신호 대비 잡음 비율(Peak noise to noise ratio: PSNR), 구조 유사성(Structural similarity: SSIM) 그리고 영상 품질 기준(Video quality metric: VQM)[50]이 있다. 그 중 PSNR 은 가장 간단하면서도 지금까지 영상 평가에 있어서, 프레임 단위로 원본 프레임과 비교하는 가장 일반적이고 객관적인 지표로 사용되었다[51]. PSNR은 평균 제곱 오차(Mean square error: MSE)를 이용한다. PSNR은 신호가 가질 수 있는 최대 전력에 대한 잡음의 전력을 나타낸 값으로, 이미지 관점에서 최대 신호 대 잡음비는 신호의 전력에 대한 고려 없이 평균 제곱의 오차를 이용해서 하기와 같이 계산될 수 있다.

$$\mathrm{PSNR} = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

$$= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

$$\mathrm{MSE} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j)-K(i,j)]^2$$

여기서 MAX_I 는 해당 이미지의 최대값으로, 해당 채널의 최대값에서 최소값을 빼서 구할 수 있다. 예를 들어 8bit 그레이스케일영상의 경우는 255(255-0)이 된다. 로그스케일에서 측정하기 때문에, 단위는 dB이며, 손실이 적을수록 높은 값을 가지며 보통 30dB가 넘으면 두 영상의 차이를 눈으로 구분 할 수 없다고 평가한다. 결과적으로 시각 시스템 (HVS)[10] 에 기초하여 이미지의 뭉개짐, 잡음, 색상의 왜곡 등 인지적 특성을 포함하여 MOS가 가능하며, 영상의 관점에서 좋은 품질로 간주 할 경우, 적어도 30dB 이하의 평균 PSNR 값을 가져야 한다. PSNR값을 기반으로 하여 MOS에 매핑한 결과를 표 6.1와 같이 표현 할 수 있다.

표 6.1 MSE/PSNR/MOS 맵핑

MSE	PSNR (dB)	MOS
< 13	> 37	Excellent
13 - 64	31 - 37	Good
65 - 205	25 - 30	Regular
206 - 650	20 - 25	Poor
> 651	< 20	Bad

애니메이션의 그래픽 영상의 성능 평가는, 단일 프레임의 비교가 아닌 영상의 일련의 흐름의 비교가 필요하다. 이 경우, 무 손실 영상의 경우에는 MSE가 0이기 때문에 PSNR은 무한대로 정의 될 수 없기 때문에, 영상에서의 화질 평가의 평균을 구할 수 없다. 그러므로 본 논문에서는 영상의 일련의 흐름을 평가함에 있어서 MSE를 통해 정량화된 수치를 기반으로 성능을 비교 분석한다.

6.2 성능 및 소모 전력 평가

6.2.1 프레임 간 변화량을 이용한 동적 렌더링 기법

6.1 절에서 정의한 것과 같이 크게 두 가지 시나리오로 분류하여 실험을 진행하였다. 먼저, 하드웨어의 성능 제약으로 디스플레이 동기화 속도(1/60초: 60fps)를 만족하지 못하는 경우의 대표 응용프로그램으로 GFXBenchmark와 Asphlat 8을 통해 성능을 평가한다. 그림 6.3 은 주어진 하드웨어에서의 원래 성능을 기준으로 정규화 하고, 제안한 프레임 간 변화량 기반 가변 해상도 기법(DRQS) [3, 6]에 따른 성능 개선을 확인한다. 추가적으로 성능의 개선과 함께 사람 시각 시스템 (HVS)에 기준에 만족하는 MSE값을 보여준다.

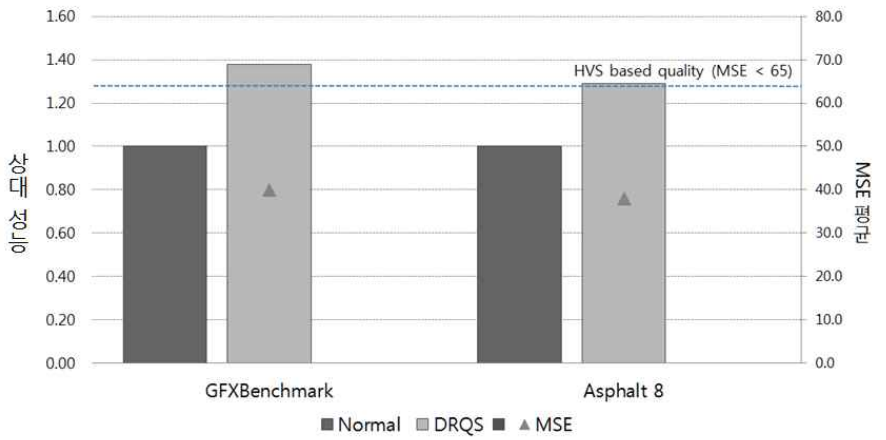


그림 6.3 HVS 관점 고 사양 그래픽 응용프로그램에서의 DRQS

주어진 GPU 요구 연산량이 매우 높은 두 개의 대표 응용 프로그램들은 빠르고 많은 객체의 이동과 빈번한 카메라 시점의 변화를 처리하기 위한 복잡한 프레그먼트 쉐이딩(Fragment shading) 연산으로 이루어져 있다. 다시 말해 두 응용 프로그램 모두 채우기-한계(Fill-bounded) 의 응용프로그램으로 실험 결과가 매우 유사함을 보여준다. 사람의 인지적 관점에서는 그래픽 품질의 감소를 인지할 수 없기 위해 HVS 기준 MSE 수치가 65이하를 만족해야 하며, DRQS의 적용이후 두 응용 프로그램의 측정된 MSE는 50이하를 유지하고 있다. 결과적으로 DRQS의 적용으로 인한 화질 열화는 사용자 관점에서 그래픽 품질의 만족 시키며, GPU의 성능 관점에서 최대 38% 까지 개선되는 것이 확인된다.

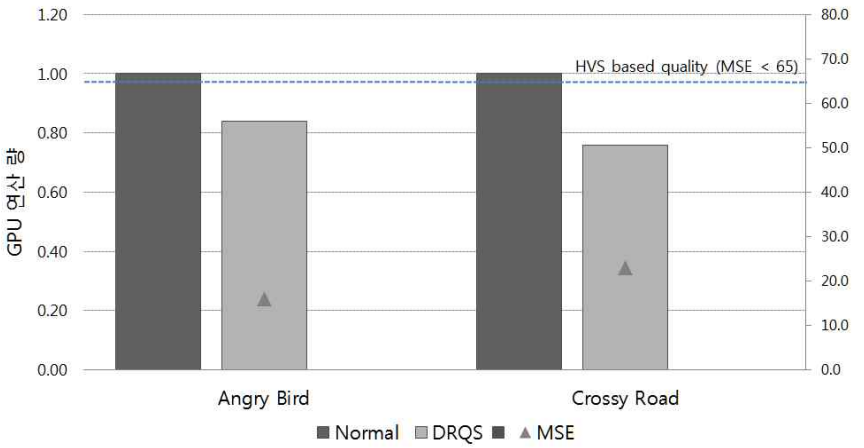


그림 6.4 HVS 관점 저 사양 그래픽 응용프로그램에서의 DRQS

반면, 그림 6.4에서는 그래픽 연산 요구량이 낮은 응용프로그램의 경우에 대한 실험 결과를 보여준다. 두 응용프로그램들은 60fps 의 이상의 성능으로 하드웨어가 렌더링이 가능한 경우로, 비록 해당 응

용프로그램들은 주어진 하드웨어 성능으로 60fps 이상 렌더링이 가능하지만, 디스플레이이의 60Hz의 제약으로 최대 1초에 60장만 렌더링 한다. 성능의 비교를 위해서 동일한 60fps를 기준으로 감소되는 GPU 연산량 결과를 보여준다. Crossy road의 경우가 Angry bird보다 프레임 간 움직임이 많기 때문에, 사람의 인지적 관점에서 DRQS는 보다 낮은 해상도로 많이 렌더링이 가능하다. 그 결과, Crossy road에서의 GPU 연산량 감소가 24%로 Angry bird의 16% 보다 높다. 당연히 Crossy road가 프레임 간 움직임이 Angry bird의 경우보다 많기 때문에 MSE 측면에서 조금 더 화질의 열화가 많이 발생할 수밖에 없다. 하지만, DRQS 적용 이후에도 두 응용 프로그램 모두 HVS 관점에서 그래픽 품질 기준을 만족하는 것을 확인할 수 있다.

그림 6.5와 그림 6.6은 각각 고 사양의 응용 프로그램과 저 사양의 응용 프로그램 4개의 원본 이미지와 DRQS의 결과 이미지를 확대비교하여 보여준다. 그림에서 보듯이 동적 렌더링으로 인한 낮은 해상도의 프레임을 스케일링 했을 경우 발생 가능한 뭉개짐 현상을 확인 할 수 있다. 이러한 스케일링으로 인한 뭉개짐을 개선하기 위해 낮은 해상도의 프레임을 렌더링 할 때 제안하는 화질 기반 렌더링을 통해 최대한 선명한 수준의 상태로 렌더링 한다. 이러한 렌더링 접근은 반대로 그림 6.5의 (c)와 같이 엘리어싱 현상이 두드러질 수 있다. 하지만, 이러한 날카로운 이미지의 낮은 해상도의 프레임은 스케일링을 통해 원본 수준의 화질로 복원 되는 것을 그림 6.5 (d) 에서와 같이 확인할 수 있다.

그림 6.5 고 사양 응용프로그램의 원본과 DRQS 비교
(a) 원본 이미지 (b) 낮은 해상도의 스케일링 (c) 화질개선 낮은 해상도 렌더링 (d) DRQS의 결과

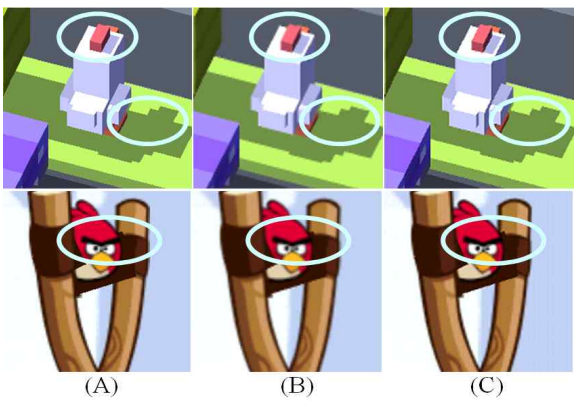


그림 6.6 저 사양 응용프로그램의 원본과 DRQS 비교
(a) 원본 이미지 (b) 낮은 해상도의 스케일링 (c) 화질개선 낮은 해상도 렌더링 (d) DRQS의 결과

마지막으로, 그림 6.7은 각각의 정의한 4가지 시나리오에서의 GPU 소모 전력의 변화와 시스템 레벨에서의 소모 전력의 변화를 통해 가변 해상도 기반 DRQS의 에너지 효율을 보여준다. 고 사양의 응용 프로그램에서는 시스템 관점에서 3%미만의 추가 소모 전류만으로 위에서 보여준 실험결과와 같이 38%렌더링 성능이 증가가 가능하다. 저 사양의 응용프로그램의 경우 이미 60fps 의 성능을 달성했기 때문에, DRQS 는 GPU는 최대 -23%, 시스템은 -18%의 에너지 감소를 보여준다.

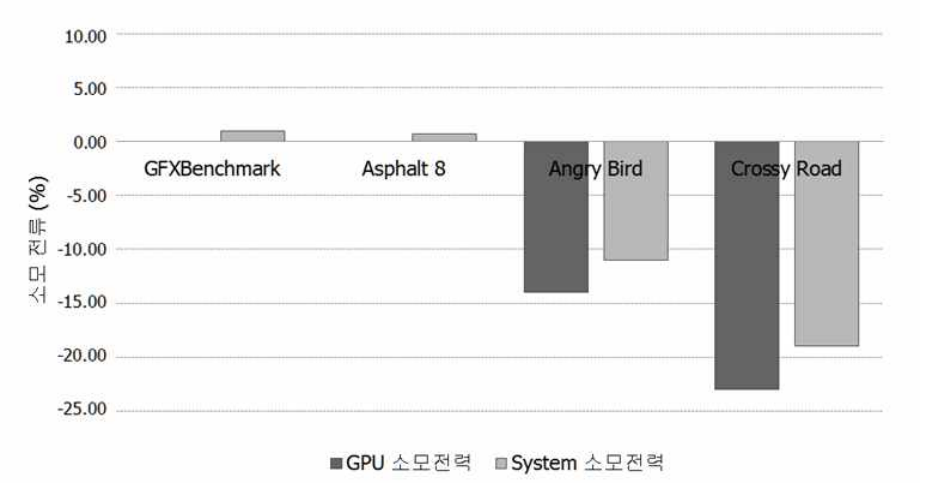


그림 6.7 DRQS의 에너지 효율 실험 결과

6.2.2 타일 기반 GPU를 위한 프레임 속도 증가 기법

6.2.1 절에서의 실험과 유사하게 타일 기반의 모바일 GPU에서 중간 프레임을 생성하기 위한 비용과 성능을 평가한다. 앞서 정의한 바와 같이 GPU의 요구 연산량이 매우 높은 경우는, 디스플레이 동기화 신호(1/60초) 기준으로 60fps을 만족하지 못하는 경우로, 주어진 대표 응용 프로그램은 GFXBenchmark와 Asphalt 8를 기반으로 주어진 GPU 하드웨어의 성능이 30fps이 구현된다고 가정하였다.

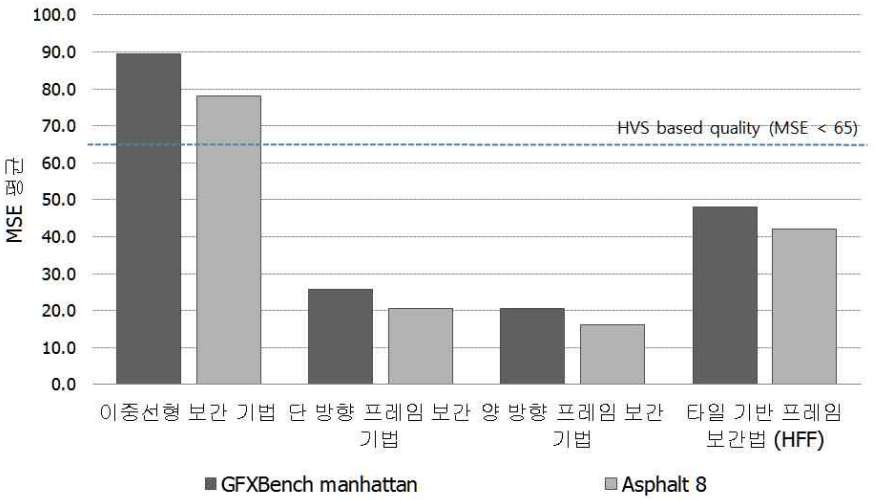


그림 6.8 최근 연구들과 HFF의 HVS 관점 성능 비교

그림 6.8은 프레임 속도를 60fps까지 올리기 위하여, 제안하는 기법과 최근 연구된 프레임 속도 증가 기법들을 가장 기본적인 이중

선형 보간(Bilinear interpolation)을 기준으로 화질과 성능 실험 결과를 보여 준다. 먼저 사람이 인지 능력(HVS) 기준에서의 최근 연구된 프레임 보간 기법들[26, 29, 32, 37]을 통한 증가 기법의 성능을 보면 MSE 30이하로서 상당히 좋은 결과를 보여주고 있으며, 제안하는 타일 기반 GPU를 위한 중간 프레임 전달 방식(Half frame forwarding: HFF) 기법[34]의 결과도 MSE 50이하를 유지하며, HVS 기준을 만족한다.

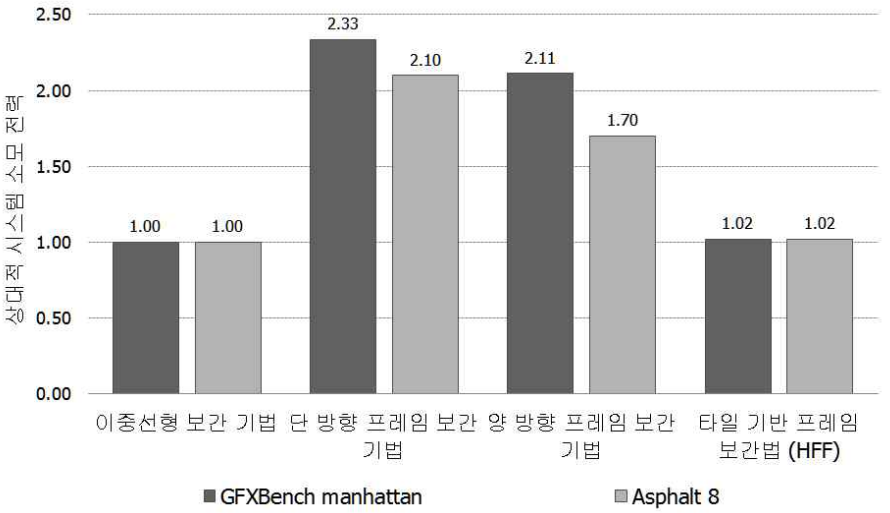


그림 6.9 프레임 보간 기법의 시스템 부하(overhead)

하지만, 그림 6.9의 시스템 소모전력 실험결과에서 보듯이, 단방향 혹은 양방향 보간 방법은 제안하는 방법은 30fps에서 60fps를 달성하기 위해 이중 보간 방법의 2배 넘는 시스템 단위의 전력을 소모한다. 최근의 프레임 보간 기법은 중간 프레임을 생성하기 위한 픽

셀을 구하기 위해 이미지 와핑을 통한 단 방향 재 투영 방법 혹은 양방향 재 투영 방법을 사용을 하며, 모션 벡터기반으로 동작하기 때문에 높은 추가 연산이 불가피 하다. GPU의 자원은 이미 하드웨어의 최대 성능을 수행하고 있기 때문에, 중간 프레임을 생성하기 위한 기존 연구들의 CPU를 통해 추가 연산이 수행된다. 언급했듯이 모든 추가적인 비용은 CA15[73]로 구성되어 있는 빅 클러스터로 수행하며, 이로 인하여 추가 연산을 위한 높은 전력 소모를 초래한다. 이러한 전력 소모는 저전력의 모바일 환경에서는 유효하지 못하다. 반면에 제안하는 타일 기반 GPU를 위한 중간 프레임 전달 방식(Half frame forwarding: HFF)은 추가 비용이 거의 없이 60fps 달성 가능함을 보여준다.

그림 6.10 HFF의 결과와 원본 이미지와의 비교
(a) 원본 이미지 (b) HFF의 결과 (c) 원본과 HFF 결과와의 차이

마지막으로, 그림 6.10에서는 원본이미지와 제안하는 타일 기반 GPU를 위한 중간 프레임 전달 방식(Half frame forwarding: HFF)를 통한 최종 이미지의 결과를 비교한다. 경계 검출을 통해 프레임

간의 동적인 영역을 포함하는 타일들을 추출하여, 추출된 타일을 우선적으로 갱신하기 때문에, 화질 열화 최소화가 가능함을 확인 할 수 있다. 추가 적으로, 3.2절에서 설명한 그림 6.11에서는 타일 간의 경계에서 발생할 수 있는 엘리어싱 현상을 타일 경계간의 보간을 통해 개선된 결과를 보여준다.

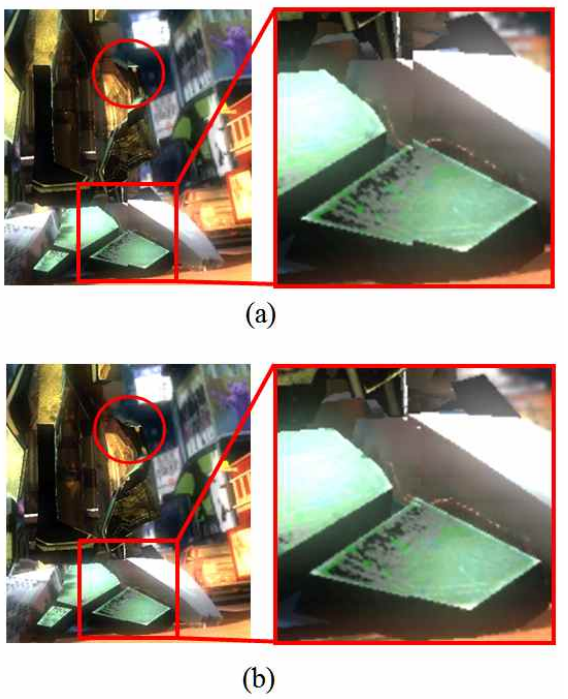


그림 6.11 타일 간 단절 현상과 경계 간 보간을 통한 개선
(a) 타일 경계 간 단절 현상 (b) 경계 간 보간을 통한 개선 결과

6.2.3 멀티 렌더 타깃을 위한 데이터 재사용 기법

동시에 하나 이상의 여러 개의 타깃을 렌더링하기 때문에 많은 메모리 대역폭을 필요로 하는 단점을 보안하기 위해, 먼저 재사용하는 렌더 타깃 개수에 따른 메모리 대역폭 감소를 그림 6.12에서 보여준다. 멀티 렌더 타깃(Multi-render target: MRT) 기술은 OpenGL ES 3.x의 발표와 함께 가장 최근에 지원을 시작하여, 대부분 현재 상용화되어 있는 응용 프로그램은 MRT를 지원하지 않는다. 그래서 실험에서는 MRT를 지원하면서, 그래픽 적으로 가장 복잡한 응용 프로그램으로 GFXBenchmark의 Manhattan 3.0[83] 기반으로 한다. 추가적으로, MRT 재사용이 주는 GPU의 쉐이더 연산의 복잡도의 변화를 확인하기 위해서 GFXBenchmark의 복잡도를 달리 한 실험 벡터를 추가적으로 구현하였다. 하나는 원본 그대로의 복잡도가 높은 벡터로 다른 하나는 연산량을 줄인 복잡도가 상대적으로 낮은 벡터로 정의한다.

그림 6.12는 사람의 인지능력 즉 시각 시스템(Human visual system: HVS) 기반에서 MRT의 재사용 정도가 미치는 영향을 보여준다. MRT의 모든 렌더 타깃 4개를 재사용하게 되면, 사람의 인지능력 관점에서 HVS 기준인 MSE 65이하를 만족하지 못한다. 그러므로 최대 3개 까지만 재사용 가능함을 알 수 있다. 여기서, 재사용의 순서는 스팩큘러(Specualar), 반사(Reflection), 노멀(Normal) 그리고

컬러(Color) 순서로 하였다. 실험 결과 메모리 사용감소 관점에서 최대 15% 가까이의 감소를 확인할 수 있으며, 응용 프로그램의 쉐이딩 연산이 복잡할수록 그 효과는 더욱 극대화 되는 것을 확인할 수 있다.

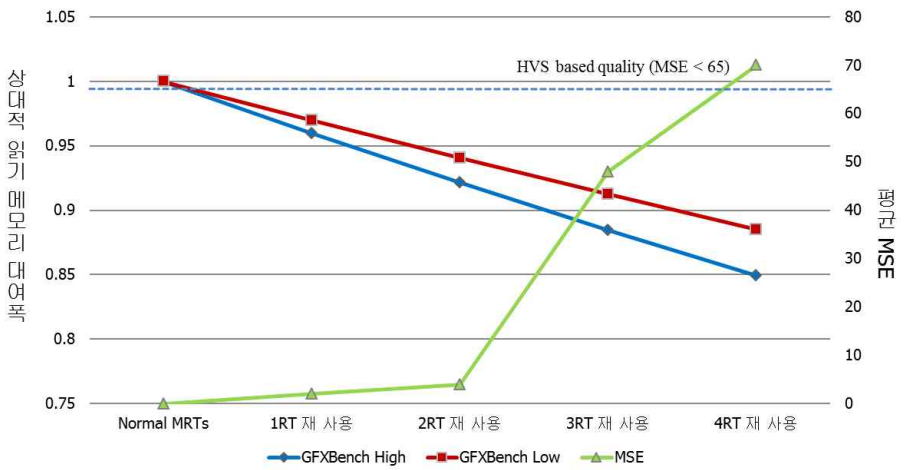


그림 6.12 HVS 기준 MRT 재사용을 통한 메모리 절감

추가적으로 그림 6.13에서는 각각의 렌더 타깃의 재사용과 화질 열화의 상관관계를 보여준다. 실험을 위해 Manhattan 3.0 실험 벡터의 모든 프레임을 30개의 대표 프레임으로 샘플링 하였다. 이 경우 렌더 타깃 중 스택큘러(Specular)와 반사(Reflection) 렌더 타깃의 정보는 재사용해도 MSE 화질 평가 기준 많은 감소를 보이지 않는다. 하지만, 컬러(Color) 렌더 타깃의 경우는 가장 많은 감소를 보이며, 여러 렌더 타깃을 동시에 재사용했을 경우의 전체 MSE의 대부분을 기여하는 것을 확인할 수 있다. 각 렌더 타깃의 재사용으로 발생할 수 있는 화질의 결점을 그림 6.14을 통해 확인 할 수 있다.

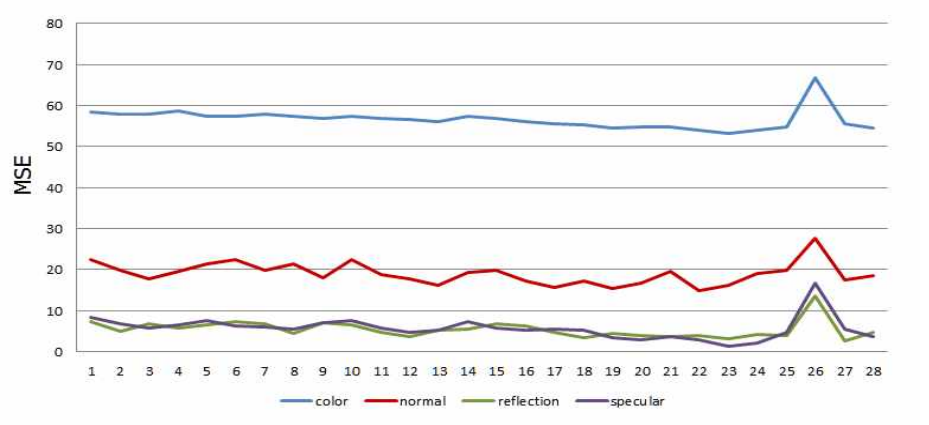


그림 6.13 재사용 렌더 타깃에 따른 화질 열화 관계

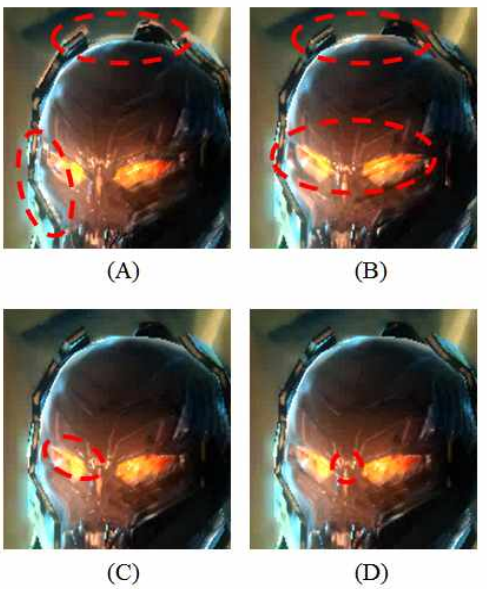


그림 6.14 렌더 타깃 재사용에 따른 결점

(a) 컬러 렌더 타깃의 재사용 (b) 노멀 렌더 타깃의 재사용

(c) 반사 렌더 타깃의 재사용 (d) 스팩큘러 렌더 타깃의 재사용

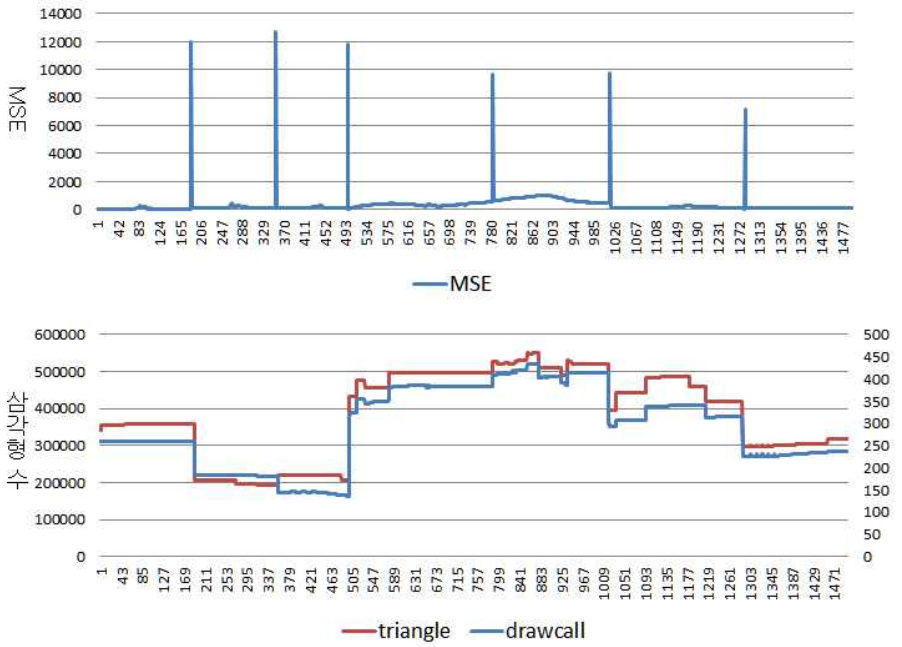


그림 6.15 프레임 구성과 드로우 콜과 정점 개수와의 관계

그림 6.15는 프레임의 흐름에서 장면 전환 시점에 발생할 수 있는 재사용의 결점과 프레임 렌더링에서 사용된 드로우 호출(Draw call)과 정점 개수와의 상관관계를 보여준다. MSE가 급격히 높아지는 시점은 프레임 장면의 전환이 일어났을 경우이며, 이 경우에 실험과 같이 삼각형의 수와 드로우 콜의 수를 통해 프레임 간 장면 전환 여부를 확인하여, MRT의 재사용을 결정하는데 이용 가능하다. 즉, 장면 전환이 많이 일어나는 구성으로 이루어진 프레임들의 경우 재사용의 기회가 줄어 그만큼 효율이 떨어지게 된다.

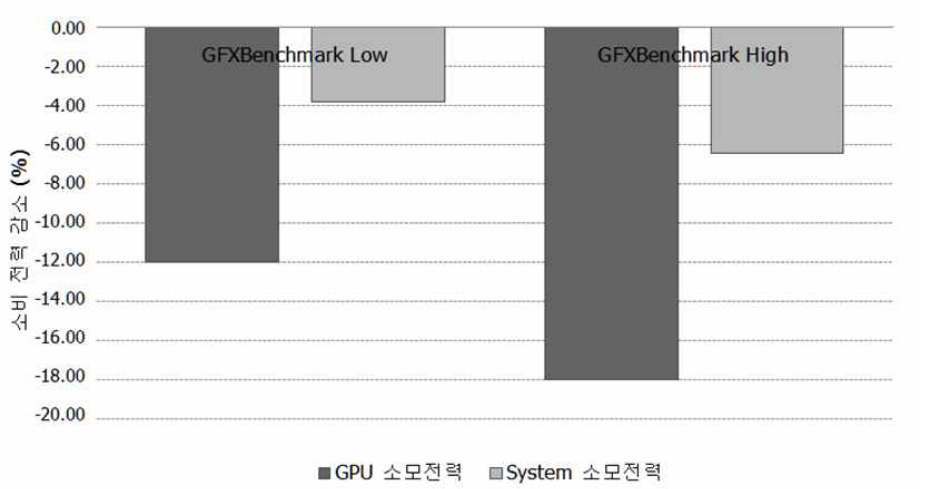


그림 6.16 MRT 재사용을 통한 소비 전력 효율

그림 6.16 은 제안하는 MRT 재사용 기법을 통해 GPU의 소모 전력과 시스템 레벨에서의 소모 전력을 보여준다. 쉐이더의 연산이 복잡하고 많을수록 MRT의 재사용을 통한 효율이 높게 나타난다. 이미 쉐이딩이 완료된 데이터의 재사용을 통해 절감되는 메모리 대역폭과 연산 절감이 복잡도가 높은 연산일수록 그 만큼 많이 되기 때문에 GPU 연산량 역시 많이 감소하게 된다. 결과적으로 GPU의 소모 전력은 GFXBenchmark Manhattan 3.0의 경우 MRT 재사용 전과 비교하여 최대 18%까지 감소되는 것을 확인할 수 있다. 결과적으로 GPU의 소모 전력 감소는 시스템 전체의 소모로 이어져 에너지 효율을 높일 수 있음을 보여준다.

표 6.2에서는 본 논문에서 제안한 사람의 인지적 능력을 고려한 그래픽 품질 개선 및 전력 소모 최적화를 위한 3가지 기법들의 성

능 및 소모 전력의 비교를 보여준다. 고 사양의 그래픽 응용 프로그램의 경우, 동일한 GPU의 전력 소모를 기반으로 최소한의 시스템의 부하로 각각의 성능 향상을 확인할 수 있다. 타일 기반 GPU를 활용한 프레임 보간 방식이 다른 2개의 접근 방법보다 성능 관점에서 높은 효율을 보인다. 비록 화질 열화가 상대적으로 가장 높지만, HVS 기준인 MSE 65이하를 만족하기 때문에 유효하다.

표 6.2 제안한 기법들의 성능 및 소모 전력 관점 비교

실험 결과		그래픽 관점 고 사양 응용 프로그램 (하드웨어 성능 < 60 fps)				그래픽 관점 저 사양 응용 프로그램 (하드웨어 성능 > 60 fps)			
		소모 전력 관점		화질 열화 (HVS<65)	**성능**	**소모 전력 관점**		화질 열화 (HVS<65)	성능
접근 방법	제안한 기법	GPU	시스템	MSE	**FPS**	GPU	**시스템**	MSE	FPS
가변 해상도 기반 최적화	DRQS	+0%	+2%	< 40	**+38%**	-23%	**-18%**	< 24	+0%
프레임 보간을 기반 프레임 속도 기반 최적화	HFF	+0%	+2%	< 48	**+50%**	-50%	**-21%**	< 30	+0%
데이터 재사용 기반 최적화	MRT 재사용	+0%	+1%	< 4	**+4%**	-20%	**-7%**	< 4	+0%

저 사양의 그래픽 응용 프로그램의 경우, GPU는 디스플레이의 60Hz보다 빠르게 렌더링이 가능하기 때문에 디스플레이의 동기화 시간(1/60초) 기준으로 최대 성능은 60fps로 고정된다. 이 경우도 제안하는 타일 기반 GPU를 활용한 프레임 보간 방식이 제안한 다른 기법들 보다 상대적으로 높은 효율을 보여준다. MRT를 활용한 데

이터 재사용은 렌더 타깃의 재사용의 최대 개수는 HVS 기준으로 정해져 있기 때문에 성능 및 소모 전력 효율성 측면에서 다른 제안한 기법들 보다 낮지만, 화질 열화 측면에서는 원본 대비 거의 화질 차이가 없는 높은 성능을 보인다.

제 7 장

결론

최근 모바일 산업의 급격한 발전으로 모바일 기기의 역할은 PC의 기능을 대신하고 있는 추세이다. 그에 따라 그래픽스 관점에서 모바일의 제한된 환경에서 사용자의 높은 그래픽 품질 요구 사항을 만족해야 하며, 실시간 처리 기준으로 빠른 사용자의 응답성이 보장되어야 한다. GPU의 소모 전력은 GPU의 연산량의 증가와 거의 정비례하기 때문에 GPU의 연산량을 줄이기 위한 다양한 연구가 되어왔다.

본 논문에서는 먼저 GPU 전력 소모를 상세히 분석 하여 전력 소모의 주요 요인으로 해상도, 프레임 속도 그리고 데이터 중복성으로 정의했다. 해상도의 증가는 GPU의 전력 소모에 매우 치명적인 영향을 끼치는 것을 확인할 수 있었다. 기존의 다양한 가변 해상도 기반 GPU 연산량을 줄이는 연구들의 분석을 통해 기존 연구들은 사람의 인지 능력을 고려가 부족하거나, 응용 프로그램의 특성만을 고려한 강제적 프레임 생략을 통해 그래픽 결점들이 존재함을 확인하였다. 이러한 문제점을 개선하기 위해 본 논문에서는 동적 렌더링 화질 개선 스케일링(Dynamic rendering quality scaling: DRQS)을 제안 하였다. DRQS는 동일한 GPU의 연산량 기반으로 최소한의 추가비용을 통해 성능을 최대 38%까지 개선하였으며, 저 사양 그래픽

응용프로그램의 경우에는 사람의 인지 능력 관점에서 화질 열화 없이 24% 까지 GPU의 연산량을 줄일 수 있는 것을 확인하였다.

둘째 요인인 프레임 속도 증가를 활용하여 그래픽 품질 개선을 고려하였다. 최근 연구들은 대부분은 프레임 보간 기반으로 모션 벡터기반 재 투영 방식을 통해 중간 프레임을 생성하기 때문에, 높은 비용이 요구된다. 이러한 비용은 모바일 환경에서 적합하지 않는 것을 확인할 수 있었다. 이러한 문제를 해결하기 위해 기존 연구 방법들과는 완전히 새로운 접근 방식인 타일 기반의 중간 프레임 전달 기법을 통한 중간 프레임을 생성 방법을 제안했다. 결과 적으로 제안하는 프레임 보간 기법은 최근 까지 연구된 프레임 보간 기법과는 다른 타일 기반 GPU를 활용한 접근 방식으로 시스템 전력 소모 측면에서 약 절반의 에너지로 동일한 성능을 발휘 할 수 있었다. 추가적으로 제안된 기법 적용 전 대비 동일한 성능 기준으로 시스템 레벨의 전력 감소를 20.4%까지 확인할 수 있었다.

셋째 주요 요인으로 데이터의 일관성을 통한 최적화 기반인 멀티 렌더 타깃(Multi-render target: MRT) 의 데이터 재사용 방법을 제안하였다. 최근 발표된 OpenGL ES 3.x 의 대표적인 기술임에도 불구하고, 기존 연구에서는 아직까지 MRT를 고려한 데이터 재사용 연구가 진행된 적이 없었다. 프레임 간 시간적 중복성을 이용하여, 동시에 다수에 쓰는 렌더링으로 인한 메모리 부하를 줄이기 위해 이전에 렌더링된 타깃의 데이터를 재사용하는 기법을 제안 하였다. 결과적으로 MRT 이용의 단점인 요구되는 매우 큰 메모리 대역폭을

해결하여 제한된 모바일 환경에 적합하도록 개선하였다. OpenGL ES 3.x가 지원되는 응용프로그램의 다양한 실험을 통해 우리는 18%의 시스템 레벨의 전력 소모 감소를 확인할 수 있었다.

위의 모든 제안 기법들의 주요 핵심은 사람의 인지 능력을 고려한 최적화 기법이다. 즉, 사람의 인지 능력 범위를 기준으로 불필요한 연산을 최소화 하여 GPU의 연산량을 줄이고, 이는 곧 시스템 전체의 전력 소모를 줄이게 되는 효과를 얻을 수 있다. 보다 정량화된 데이터 기반으로 분석하기 위해 MOS(Mean opinion score) 맵핑 기반 MSE(Mean square error) 값을 정의하였으며, 이 값을 기반으로 인간 시각 시스템(Human-Visual System: HVS)의 기준을 정의할 수 있었다. 이렇게 정의된 기준은 GPU 연산량 감소의 임계치를 활용하여 사람의 그래픽 품질 감소를 인지하지 못하는 범위 내에서만 연산량 최적화가 이루어졌다. 이를 통해 제안된 3 가지의 주요 기법들은 모바일 환경에서 보다 높은 사용자의 기대 수준을 만족할 수 있음을 증명하였고, 동일한 연산량 기반으로 높은 응답성을 보장하기 위해 디스플레이의 고정된 높은 프레임 속도를 만족할 수 있었다. 또한. 데이터 재사용을 통한 감소된 연산량으로 높은 그래픽 렌더링 효과를 구현할 수 있었다. 결과적으로 위의 연구 성과를 통해 모바일의 제한된 환경에서 높은 품질의 그래픽 서비스를 제공함과 동시에 GPU의 전력 소모 관점에서 최대 50%의 감소와 시스템 관점에서의 최대 21%의 감소에 기여하였다.

참고 문헌

[1] DelBarrio, V. M., González, C., Roca, J., Fernández, A., and Espasa, R. "ATTILA: a cycle-level execution-driven simulator for modern GPU architectures." In *Proceeding of Performance Analysis of Systems and Software,* IEEE, 2006. pp. 231-241

[2] Seshadrinathan, K., Soundararajan, R., Bovik, A. C., and Cormack, L. K. "Study of subjective and objective quality assessment of video." *Image Processing IEEE transactions on, 19*(6). IEEE, 2010. pp. 1427-1441.

[3] Kim, M. K., Ki, S. H., Seo, Y. D., Park, J. H., and Jeon, C. S. "Dynamic Rendering Quality Scaling based on Resolution Changes." *IEICE Transactions on Information and Systems, 8*(12). IEICE, 2015. pp. 2353-2357.

[4] Williams, L. "Casting curved shadows on curved surfaces." *ACM Siggraph Computer Graphics, 12*(3). ACM, 1978. pp. 203-220.

[5] Cook, R. L., and Kenneth, E. T. "A reflectance model for computer graphics." *ACM Transactions on Graphics (TOG), 1*(1). ACM, 1982. pp. 7-24

[6] Kim, M. K., Ki, S. H., Seo, Y. D., Shin, C. H., and Park, J. H. "Dynamic Rendering Quality Scaling for Mobile GPU" In *Proceedings of ACM Siggraph Asia 2015.* ACM, 2015.

[7] Reeves, W. T. "Particle systems—a technique for modeling a class of fuzzy objects." *ACM Transactions on Graphics (TOG), 2*(2). ACM, 1983. pp. 91-108

[8] Akenine-Möller, T., and Ström, J. "Graphics processing units for handhelds." In *Proceedings of the IEEE, 96*(5). IEEE, 2008. pp. 779-789.

[9] Ström, J., and Akenine-Möller, T. "i-PACKMAN: High quality, low complexity texture compression for mobile phones." In *Proceedings of the ACM Siggraph/Eurographics conference on Graphics hardware*. ACM, 2005. pp. 63-70

[10] Zeadally, S., Cerqueira, E., Curado, M., and Leszczuk, M. *Future multimedia networking*. Springer-Verlag Berlin Heidelberg, 2009. pp. 1-13.

[11] Mochocki, B., Kanishka L., and Srihari C. "Power analysis of mobile 3D graphics." In *Proceedings of the conference on Design, automation and test in Europe: Proceedings*. European Design and Automation Association, 2006. pp. 502-507.

[12] Herzog, R., Eisemann, E., Myszkowski, K., and Seidel, H. P. "Spatio-temporal upsampling on the GPU." In *Proceedings of the 2010 ACM Siggraph symposium on Interactive 3D Graphics and Games*. ACM, 2010. pp. 91-98.

[13] IDC. Android and iOS Continue to Dominate the Worldwide Smartphone Market. Retrieved Feb, 2015. http://www.idc.com/getdoc.jsp?containerId=prUS24676414

[14] Shinya, M. "Improvements on the Pixel-tracing Filter: Reflection/Refraction, Shadows & Jittering." In *Proceeding of Graphics Interface*. Canadian Information Processing Society, 1995. pp. 92-98.

[15] Tomasi, C., and Roberto M. "Bilateral filtering for gray and color images." *Computer Vision, 1998. Sixth International Conference on*. IEEE, 1998. pp. 839-846.

[16] Smith, S. M., and Brady, J. M. "SUSAN－A new approach to low level image processing." *International journal of computer vision, 23*(1). 1997. pp. 45-78

[17] Eisemann, E., and Frédo D. "Flash photography enhancement via intrinsic relighting." *ACM transactions on graphics (TOG), 23*(3). ACM, 2004. pp. 673-678.

[18] Petschnigg, G., Szeliski, R., Agrawala, M., Cohen, M., Hoppe, H., and Toyama, K. "Digital photography with flash and no-flash image pairs." *ACM transactions on graphics (TOG), 23*(3). ACM, 2004. pp. 664-672.

[19] Kopf, J., Cohen, M. F., Lischinski, D., and Uyttendaele, M. "Joint bilateral upsampling." *ACM Transactions on Graphics (TOG), 26*(3). ACM, 2007. pp. 96-101

[20] Yang, L., Pedro V. S., and Jason L. "Geometry Aware Framebuffer Level of Detail." *Computer Graphics Forum, 27*(4). Blackwell Publishing Ltd, 2008. pp. 1183-1188.

[21] He, S., Liu, Y., and Zhou, H. "Optimizing Smartphone Power Consumption through Dynamic Resolution Scaling." In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 2015. pp. 27-39.

[22] Akyüz, A. O. "High dynamic range imaging pipeline on the gpu." *Journal of Real-Time Image Processing, 10*(2). 2012. pp. 273-287.

[23] Bavoil, L., "Advanced soft shadow mapping techniques." In *Proceeding of the game developers conference*. 2008. pp. 11-17.

[24] Yu, T. T., "Depth of field implementation with OpenGL." *Journal of Computing Sciences in Colleges, 20*(1). 2004. pp. 136-146.

[25] LucidLogix, PowerXtend Products, Retrieved February, 2015. http://lucidlogix .com/products/powerxtend-for-android

[26] Didyk, P., Eisemann, E., Ritschel, T., Myszkowski, K., and Seidel, H. P. "Perceptually-motivated Real-time Temporal Upsampling of 3D Content for High refresh rate Displays. " *Computer Graphics Forum, 29*(2). Blackwell Publishing Ltd, 2010. pp. 713-722.

[27] Didyk, P., Ritschel, T., Eisemann, E., Myszkowski, K., and Seidel, H. P. "Adaptive Image-space Stereo View Synthesis." In *Proceeding of VMV.* 2010. pp. 299-306.

[28] Yu, X., Wang, R., and Yu, J. "Real-time Depth of Field Rendering via Dynamic Light Field Generation and Filtering." *Computer Graphics Forum, 29*(7). Blackwell Publishing Ltd, 2010. pp. 2099-2107.

[29] Yang, L., Tse, Y. C., Sander, P. V., Lawrence, J., Nehab, D., Hoppe, H., and Wilkins, C. L. "Image-based bidirectional scene reprojection." *ACM Transactions on Graphics (TOG), 30*(6). ACM, 2011. pp. 150-160.

[30] Nehab, D., Sander, P. V., Lawrence, J., Tatarchuk, N., and Isidoro, J. R. "Accelerating real-time shading with reverse reprojection caching." In *Proceeding of Graphics hardware*, 2007. pp. 61-67.

[31] Scherzer, D., Jeschke, S., and Wimmer, M. "Pixel-correct shadow maps with temporal reprojection and shadow test confidence." In *Proceedings of the 18th Eurographics conference on Rendering Techniques*. Eurographics Association, 2007. pp. 45-50.

[32] Andreev, D. "Real-time frame rate up-conversion for video games: or how to get from 30 to 60 fps for free." In *Proceedings of ACM Siggraph 2010*. ACM, 2010. pp. 16.

[33] Klompenhouwer, M. A., and Velthoven, L. J. "Motion blur reduction for liquid crystal displays: motion-compensated inverse filtering." In *Proceeding of Electronic Imaging 2004*. International Society for Optics and Photonics, 2004. pp. 690-699.

[34] Park, J. H., Kim M. K., Ki, S. H., Seo, Y. D. and Shin, C. H. "Half Frame Forwarding:Frame-rate Up Conversion for Tiled Rendering GPU." In *Proceedings of ACM Siggraph 2015.* ACM, 2015.

[35] Pan, H., Feng, X. F., and Daly, S. "LCD motion blur modeling and analysis." In *Proceeding of Image Processing, ICIP 2005.* IEEE, 2005. pp. 11-21.

[36] Takeuchi, T., and De Valois, K. K. "Sharpening image motion based on the spatio-temporal characteristics of human vision." In *Proceeding of Electronic Imaging 2005*. International Society for Optics and Photonics, 2005. pp. 83-94.

[37] Bowles, H., Mitchell, K., Sumner, R. W., Moore, J., and Gross, M. "Iterative image warping." *Computer graphics forum, 31*(2). Blackwell Publishing Ltd, 2012. pp. 237-246.

[38] Scherzer, D., Schwärzler, M., Mattausch, O., and Wimmer, M. "Real-time soft shadows using temporal coherence." In *Proceeding of Advances in Visual Computing*. Springer Berlin Heidelberg, 2009. pp. 13-24.

[39] Yang, L., Nehab, D., Sander, P. V., Sitthi-amorn, P., Lawrence, J., and Hoppe, H. "Amortized super sampling." *ACM Transactions on Graphics (TOG), 28*(5). ACM, 2009. pp. 135-147.

[40] Mattausch, O., Scherzer, D., and Wimmer, M. "High-Quality Screen-Space Ambient Occlusion using Temporal Coherence." *Computer Graphics Forum, 29*(8). Blackwell Publishing Ltd, 2010. pp. 2492-2503.

[41] McReynolds, T., and Blythe, D. *Advanced graphics programming using OpenGL*." Elsevier, 2005. pp. 312-315.

[42] Sharma, P. "Challenges with virtual reality on mobile devices." In *Proceedings of ACM Siggraph 2015*. ACM, 2015. pp. 57.

[43] Fenney, S. "Texture compression using low-frequency signal modulation." In *Proceedings of the ACM Siggraph/Eurographics conference on Graphics hardware*. Eurographics Association, 2003. pp. 84-91.

[44] Ström, J., and Pettersson, M. "ETC 2: texture compression using invalid combinations." In *Proceeding of Graphics Hardware*. 2007. pp. 49-54.

[45] Ki, S. H., Park, J. H., Nah, J. H., Kim, M. K., Seo, Y. D., and Shin, C. H. "Reusing MRTs for Mobile GPUs." In *Proceedings of ACM Siggraph Asia 2015*. ACM, 2015.

[46] Antochi, I. *Suitability of tile-based rendering for low-power 3d graphics accelerators*. TU Delft, Delft University of Technology, 2007. pp. 83-92.

[47] Cox, M., and Hanrahan, P. "Pixel merging for object-parallel rendering: a distributed snooping algorithm." In *Proceeding of Parallel Rendering Symposium, 1993*. IEEE, 1993. pp. 49-56.

[48] Antochi, I., Juurlink, B., Vassiliadis, S., and Liuha, P. "Memory bandwidth requirements of tile-based rendering." In *Proceeding of Computer Systems: Architectures, Modeling, and Simulation*. Springer Berlin Heidelberg, 2004. pp. 323-332.

[49] Seshadrinathan, K., Soundararajan, R., Bovik, A. C., and Cormack, L. K. "Study of subjective and objective quality assessment of video." *Image Processing, IEEE transactions on, 19*(6). IEEE, 2010. pp. 1427-1441.

[50] Van den Branden Lambrecht, C. J., and Verscheure, O. "Perceptual quality measure using a spatiotemporal model of the human visual system." In *Proceeding of Electronic Imaging: Science &Technology*. International Society for Optics and Photonics, 1996. pp. 450-461.

[51] Wang, Z., Lu, L., and Bovik, A. C. "Video quality assessment based on structural distortion measurement." *Signal processing: Image communication, 19*(2). 2004. pp. 121-132.

[52] Shye, A., Scholbrock, B., and Memik, G. "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures." In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009. pp. 168-178.

[53] Qualcomm, Adreno Wikipedia. Retrieved Nov, 2014. http://en.wikipedia.org/wiki/Adreno

[54] Qualcomm, FlexRender. Retrieved Nov, 2014. http://www.qualcomm.com/media/videos/flexrender-rendered-useful

[55] Qualcomm. Composition with Snapdragon. Retrieved Nov, 2014. http://developer.qualcomm.com/sites/default/files/composition-with-snapdragon.pdf.

[56] Nvidia. Tegra. Retrieved Nov, 2014. http://www.nvidia.com/object/tegra.html

[57] Nvidia. Tegra 4 Family GPU Architecture. Retrieved Nov, 2014. http://www.nvidia.com/docs/IO//116757/Tegra_4_GPU_ Whitepaper_FINALv2.pdf.

[58] Nvidia. Hardware Occlusion Queries Made Useful. Retrieved Nov, 2014. http://developer.nvidia.com/GPUGems2/gpugems2_chapter06.html.

[59] Google. Android SDK. Retrieved Jan, 2015. http://developer.
.android.com/sdk/index.html
[60] Google. Android Graphics. Retrieved Jan, 2015. http://source
.android.com/devices/graphics.html.
[61] Apple, iOS Dev Center. Retrieved Jan, 2015. http://develope
r.apple.com/devcenter/ios/index.action.
[62] Khronos Group Inc. OpenGL ES. Retrieved Dec, 2014. http:
//www.khronos.org/opengles/
[63] Khronos Group Inc. OpenGL. Transform Feedback. Retrieved
Feb, 2015. https://www.opengl.org/wiki/Transform_Feedback
[64] Khronos Group Inc. OpenGL. Retrieved Feb, 2015. Multiple
Render Targets Wikipedia. http://en.wikipedia.org/wiki/Mul
tiple_Render_Targets
[65] Khronos Group Inc. OpenGL. Compute Shader. Retrieved
Feb, 2015. https://www.opengl.org/wiki/Compute_Shader
[66] LG Electronics. LG G3. Wikipedia. Retrieved Mar, 2015
https://ko.wikipedia.org/wiki/LG_G3
[67] LG Electronics. LG G3 Screen Wikipedia. Retrieved Mar,
2015. https://ko.wikipedia.org/wiki/LG_G3_Screen
[68] Samsung Electronics, Samsung Galaxy S6 and S6 Edge
Wikipedia. Retrieved Mar, 2015. http://en.wikipedia.org/wiki
/Samsung_Galaxy_S6_and_S6_Edge
[69] ARM, ARM flagship mobile GPU, Wikipedia. Retrieved Mar,
2015. http://en.wikipedia.org/wiki/Mali_%28GPU%29.
[70] ARM. ARM Mali-55. Retrieved Oct, 2014. http://mobile.arm.
com/products/multimedia/mali-graphics-hardware/mali-55.ph
p
[71] ARM. ARM Mali-T678. Retrieved Dec, 2014. http://www.ar
m.com/products/multimedia/mali-graphics-plus-gpu-compute
/mali-t678.php
[72] ARM. ARM Mali-T880. Retrieved Mar, 2015. http://www.ar
m.com/products/multimedia/mali-performance-efficient-graph
ics/mali-t880.php
[73] ARM. ARM Cortex-A15 Wikipedia. Retrieved June, 2014.
https://en.wikipedia.org/wiki/ARM_Cortex-A15

[74] ARM. ARM Cortex-A7 Wikipedia. Retrieved June, 2014. https://en.wikipedia.org/wiki/ARM_Cortex-A7
[75] ARM. ARM big.LITTLE Technology. Retrieved June, 2014. https://www.arm.com/products/.../biglittleprocessing.php
[76] ARM. Mali GPU Application Optimization Guide. Retrieved Dec, 2014. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0555a/CHDIAHCC.html
[77] Imagination. PowerVR. Wikipedia. Retrieved Feb, 2015. http://en.wikipedia.org/wiki/PowerVR.
[78] Imagination. PowerVR Rogue Imagination Technologies. Retrieved Feb, 2015. http://imgtec.com/powervr/graphics/rogue/
[79] National Instruments Inc. NI USB-6366. Retrieved Jan, 2015. http://sine.ni.com/nips/cds/view/p/lang/en/nid/209076
[80] Gameloft. Asphalt 8: Airborne HD. Retrieved Feb, 2015. http://www.gameloft.com/android-games/asphalt-8-free
[81] Rovio. Angry Birds. Retrieved Feb, 2015. http://www.angrybirds.com
[82] Yodol. Crossy Road. Retrieved Feb, 2015. http://www.crossyroad.com
[83] Kishonti Infomatics, GFXBench Unified Graphics Benchmark. Retrieved Feb, 2015. http://gfxbench.con/result.jsp
[84] Vivante Coporation. OpenGL ES 3.0 GPU IP Cores. Composition Processing Cores. Retrieved Feb, 2015. http://www.vivantecorp.com/index.php/en/technology/composition.html
[85] CineFX 3.0. Nvidia Technical Brief. Retrieved Feb, 2015. http://www.all4chip.com/www/APMSOFT/news/

Abstract

Although mobile GPU hardware has evolved remarkably, it does not yet satisfy the ever increasing market demand for high-quality graphics, while maintaining a consistent 60fps. Also, as resolution exponentially increases in mobile platforms, maintaining acceptable frame rate becomes more challenging than ever because elevated computing demand results in reduction of battery use time and increase in device surface temperature. Because power consumption increases almost linearly with GPU workload, the heavy GPU workload imposed by fixed high resolution and frame rate can be allowed to be actually computed only when there are human perceptible benefits.

In this thesis, we propose novel techniques to reduce GPU workload considering human perception capability. As the initial step in reducing it, we explore main sources of power drain with one of commercial smart phones LG G3. We found that three main sources can be exploited to reduce GPU workload effectively as the following: resolution, frame rate and data redundancy. Firstly we focus on resolution changes in mobile GPU. From the recent resolution change techniques, we still observe noticeable artifacts because they did not carefully investigate whether resulting sequences of frames are perceived as containing artifacts or not. Unlike the previous techniques, we propose a dynamic rendering quality scaling(DRQS) which is based on resolution changes and quality scaling to improve the performance up to 38% with a negligible overhead using a transform matrix. Further the

proposed technique can be exploited to reduces the workload up to 24% without human visual-perceptual changes. Secondly, we investigate increasing frame rate to remedy flickering artifacts exploiting interpolation techniques. The most recent techniques are not suitable for mobile devices because they have been mainly studied to motion-compensate approach and high computational costs are inevitable. To address this problem, we propose a novel frame rate up conversion based on a half frame forwarding(HFF) approach for tile-based GPU rendering, which can generate intermediate frames with a half of computational cost compared to the previous techniques. Lastly, we propose data reuse technique for multi-render target(MRT) which the most recent rendering technique proposed in OpenGL ES 3.0. MRTs enables deferred shading for rendering complex lighting operations efficiently. However, MRTs requires a huge amount of memory bandwidth and this is an obstacle for mobile devices because it facilitates rendering multiple render-target textures at once. To alleviate this problem, exploiting temporal coherence, when specific render-targets (RTs) have temporal coherence between consecutive frames, we selectively reuse RTs data. From our experiments, the performance result demonstrates that the proposed algorithm can reduce the system power consumption by up to 18% maintaining graphic quality based on human perception capability.

Keywords : GPU real-time rendering, mobile GPU, frame rate up conversion, dynamic rendering, multi-render target, date reuse, GPU power optimization, quality scaling.